

普通高等教育“十一五”国家级规划教材
高等学校规划教材·上海市精品课程教材

计算机系统结构

（第3版）

徐炜民 严允中 编著
孙德文 主审

電子工業出版社

Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

本书是普通高等教育“十一五”国家级规划教材。全书共 11 章。第 1 章介绍计算机系统结构的基本概念,以及计算机系统结构的形成和发展过程;第 2~9 章以现代计算机系统结构和并行处理为主线,对计算机系统结构的合成、存储系统结构、流水线结构、并行处理机、多处理机系统、RISC 结构、分布计算环境结构和数据流计算机结构等进行了深入的分析和探讨;第 10 章讨论软件对计算机系统结构的影响;第 11 章就现代计算机系统结构的最新发展进行了综述。本书为任课老师免费提供电子课件和例题及习题参考解答。

本书是高等学校计算机专业本科生“计算机系统结构”课程的通用教材,也可作为有关专业研究生教材和科技工作者的参考书。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有,侵权必究。

图书在版编目(CIP)数据

计算机系统结构/徐炜民,严允中编著. —3 版. —北京:电子工业出版社,2010.3

高等学校规划教材

ISBN 978-7-121-10228-8

I. 计… II. ①徐…②严… III. 计算机体系结构—高等学校—教材 IV. TP303

中国版本图书馆 CIP 数据核字(2010)第 007659 号

策划编辑:童占梅

责任编辑:童占梅

印 刷: 北京京科印刷有限公司

装 订:

出版发行:电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本: 787×1 092 1/16 印张: 22.5 字数: 572 千字

印 次: 2010 年 3 月第 1 次印刷

印 数: 4 000 册 定价: 29.80 元

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话:(010) 88254888。

质量投诉请发邮件至 zltz@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线:(010) 88258888。

前 言

本书是普通高等教育“十一五”国家级规划教材，也是上海市精品课程建设的成果。本次修订在电子工业出版社1997年出版的“九五”国家级规划教材《计算机系统结构》和2003年7月出版的“十五”国家级规划教材《计算机系统结构（第2版）》的基础上，根据最新的计算机学科教学计划和教育部普通高等教育“十一五”国家级规划教材的要求，吸取了计算机系统结构和并行处理技术发展的新理论、新成果以及国外同类教材的优点，对原教材（第2版）进行了较大修改和增补。希望通过本教材（第3版）的出版使用，听取各方面意见，为进一步提高教材质量并使之成为精品教材打下良好的基础。

本教材共分11章，参考教学时数为40~80。

第1章简要介绍计算机系统结构的基本概念，以及计算机系统结构的形成和发展过程。

第2~9章以现代计算机系统结构和并行处理为主线，本着计算机系统结构中硬中有软、软中有硬、相互转换、彼此渗透的观点，从原理、结构、分析、设计和实现等方面，对计算机系统结构的合成、存储系统结构、流水线结构、并行处理机、多处理机系统、RISC结构、分布计算环境结构和数据流计算机结构等进行了比较深入的分析和探讨。

第10章就软件对计算机系统结构的影响进行了专门的讨论与分析。

第11章就现代计算机系统结构的最新发展进行了综述。

本教材在编写过程中，力求文字精练，内容具体、准确，并能反映现代计算机系统结构的发展水平。与第2版相比，主要有如下特点：

第一，反映了国内外计算机系统结构方面比较成熟的研究成果和最新发展；

第二，内容更加全面，增加了较多的新内容，凡属于计算机系统结构的主要内容均有涉及；

第三，对许多关键知识点的陈述更加深入，本书吸取了国外同类教材部分量化研究方法，力求从系统定量分析和系统设计的高度来介绍计算机系统的基本概念和基本分析方法。

本书各章内容相对独立，使用时可以根据不同要求任选其中的部分章节组织教学。本教材将为任课教师免费提供电子课件、例题和习题参考答案。需要者可登录华信教育资源网<http://www.hxedu.com.cn> 免费注册下载。

本教材由徐炜民教授统稿，第1、2、3、4、5、9章及第6、10章的部分内容由严允中教授编写；第7、8、11章及第6、10章的部分内容由徐炜民教授编写。张吉锋教授参加了编写大纲的拟定，并对全书结构与章节的组织提出了宝贵意见。全书由上海交通大学孙德文教授担任主审，他详细校阅了全部书稿，并提出了很多具体的修改意见。在教材编写过程中，还得到了北京航空航天大学杨文龙教授的具体指导，并提出了宝贵的意见。在此向他们表示诚挚的感谢。

由于作者水平有限，书中难免存在一些缺点和错误，殷切希望广大读者批评指正。我们会在本教材重印时，及时改正。作者 E-mail: wmxu@staff.shu.edu.cn。

编著者

目 录

第 1 章 计算机系统结构导论	(1)	2.2 总线结构	(40)
1.1 计算机系统的基本概念	(1)	2.2.1 总线的分类	(41)
1.2 计算机系统的发展	(1)	2.2.2 总线结构的特点	(41)
1.2.1 冯·诺依曼体系结构的特点	(2)	2.2.3 总线通信方式	(41)
1.2.2 器件发展对系统结构的影响	(3)	2.2.4 总线仲裁	(45)
1.2.3 应用对系统结构的影响	(5)	2.2.5 总线标准	(47)
1.2.4 算法对系统结构的影响	(6)	2.3 存储系统概述	(54)
1.2.5 价格对系统结构的影响	(7)	2.3.1 存储器容量、速度与价格 的关系	(55)
1.2.6 现代计算机系统的分类和 发展过程	(8)	2.3.2 存储系统的层次结构	(56)
1.3 计算机系统的功能和结构	(8)	2.3.3 存储系统的性能参数	(58)
1.3.1 计算机系统的层次结构	(8)	2.3.4 程序访问的局部性	(60)
1.3.2 计算机系统结构定义	(11)	2.4 输入/输出系统	(62)
1.3.3 计算机组成与实现	(12)	2.4.1 输入系统	(62)
1.3.4 计算机系统结构、组成和 实现三者的关系	(12)	2.4.2 输出系统	(63)
1.3.5 计算机系统的特性	(13)	2.4.3 中断系统	(65)
1.4 计算机系统设计的方法	(15)	2.4.4 通道处理机和 I/O 处理机 ..	(67)
1.4.1 软、硬件取舍的基本原则 ..	(15)	第 3 章 存储系统结构	(74)
1.4.2 计算机系统设计定量 原则	(16)	3.1 地址映像与变换	(74)
1.4.3 计算机系统的设计任务	(19)	3.1.1 程序的定位	(74)
1.4.4 计算机系统的设计步骤	(20)	3.1.2 全相联映像及其变换	(79)
1.5 现代计算机系统结构的研究领域 ..	(21)	3.1.3 直接映像及其变换	(80)
1.5.1 计算机系统结构分类	(21)	3.1.4 组相联映像及其变换	(81)
1.5.2 现代计算机系统结构 研究方向	(24)	3.1.5 段相联映像及其变换	(82)
1.5.3 计算机系统结构发展趋势 ..	(25)	3.1.6 位选择组相联映像及其 变换	(85)
第 2 章 计算机系统结构的合成	(29)	3.1.7 对标志表的分析	(86)
2.1 中央处理器	(29)	3.1.8 散列概念在地址变换中 的应用	(86)
2.1.1 CPU 组成	(29)	3.2 替换算法及其实现	(88)
2.1.2 数据表示	(30)	3.2.1 替换算法的分析	(88)
2.1.3 寻址方式分析	(36)	3.2.2 LRU 替换算法的实现	(91)
2.1.4 指令优化	(38)	3.3 并行主存系统	(94)
		3.3.1 并行主存系统频宽分析	(94)
		3.3.2 单体多字存储器	(96)

3.3.3	多体交叉存储器	(96)	4.3.3	流水线调度优化	(152)
3.3.4	并行主存系统	(98)	4.4	流水线相关处理	(154)
3.4	高速缓冲存储器 (Cache)	(101)	4.4.1	局部相关及处理	(154)
3.4.1	Cache 基本结构和工作原理	(101)	4.4.2	全局相关及处理	(156)
3.4.2	Cache 的替换算法分析	(103)	4.4.3	流水线中断处理	(156)
3.4.3	Cache 的透明性	(105)	4.5	向量的流水处理和向量处理机	(156)
3.4.4	任务切换对失效率的影响	(106)	4.5.1	向量处理基本概念	(156)
3.4.5	多处理机系统的 Cache 结构	(106)	4.5.2	向量处理机的结构	(159)
3.4.6	“Cache-主存”层次性能分析	(107)	4.5.3	提高向量处理机性能的方法	(162)
3.4.7	Cache 性能计算	(109)	4.5.4	向量处理机的技术指标	(167)
3.5	虚拟存储器	(114)	4.5.5	多向量多处理机概述	(170)
3.5.1	虚拟存储器基本结构和工作原理	(114)	4.6	超级流水处理机	(177)
3.5.2	虚地址和辅存实地址的变换	(115)	4.6.1	超标量处理机	(177)
3.5.3	多用户虚拟存储器	(116)	4.6.2	超流水线处理机	(181)
3.5.4	加快地址变换的方法	(120)	4.6.3	超长指令字处理机	(182)
3.5.5	虚拟存储器性能分析	(122)	4.6.4	超标量超流水 VLIW 处理机	(186)
3.6	主存保护与控制	(125)	4.6.5	P6 微结构	(191)
3.6.1	主存保护	(125)	第 5 章	并行处理机	(194)
3.6.2	主存控制部件	(128)	5.1	系统结构中的并行性概念	(194)
3.6.3	磁盘冗余阵列	(129)	5.1.1	并行性概念	(194)
第 4 章	流水线结构	(133)	5.1.2	并行处理的发展	(195)
4.1	流水线结构原理	(133)	5.2	并行处理机基本结构	(199)
4.1.1	重叠方式	(133)	5.2.1	分布式存储器结构	(200)
4.1.2	先行控制	(136)	5.2.2	共享式存储器结构	(201)
4.1.3	流水线处理机	(138)	5.2.3	并行处理机特点	(202)
4.2	线性流水线技术指标	(142)	5.3	并行处理机互连网络	(203)
4.2.1	吞吐率	(142)	5.3.1	互连网络基本概念	(203)
4.2.2	加速比	(143)	5.3.2	单级互连函数	(204)
4.2.3	效率	(143)	5.3.3	互连网络特性	(209)
4.2.4	流水线段数选择	(144)	5.3.4	静态互连网络	(211)
4.3	非线性流水线处理机	(147)	5.3.5	动态互连网络	(215)
4.3.1	预约表和等待时间分析	(147)	5.3.6	多级互连网络	(217)
4.3.2	无冲突调度	(150)	5.3.7	互连网络寻径	(227)
			5.4	阵列处理机	(232)
			5.4.1	阵列处理机结构	(233)
			5.4.2	阵列处理机算法	(237)
			5.4.3	阵列处理机举例	(239)

5.5	相联处理机.....	(249)	第 8 章	分布计算环境结构.....	(298)
5.5.1	相联处理机结构.....	(249)	8.1	分布计算环境的发展.....	(298)
5.5.2	相联检索算法.....	(251)	8.2	客户-服务器结构.....	(298)
5.5.3	相联处理机举例.....	(253)	8.2.1	客户-服务器结构的特点..	(298)
第 6 章	多处理机系统.....	(256)	8.2.2	中间件的概念和特点.....	(299)
6.1	多处理机的概念.....	(256)	8.3	开放式分布处理.....	(299)
6.1.1	多处理机系统的定义.....	(256)	8.4	公共对象请求代理体系结构.....	(300)
6.1.2	多重处理对处理机特性 的要求.....	(257)	8.5	基于 Web 的分布计算.....	(301)
6.2	多处理机结构.....	(258)	8.5.1	Browser/Server 结构.....	(301)
6.2.1	多处理机的基本结构.....	(258)	8.5.2	基于 Web 的页面描述语言 标准 XML.....	(302)
6.2.2	多处理机的互连网络.....	(259)	第 9 章	数据流计算机结构.....	(304)
6.2.3	多处理机系统的存储器 结构.....	(266)	9.1	数据流计算机的基本原理.....	(304)
6.2.4	多处理机系统的特点.....	(268)	9.2	数据流计算机的指令.....	(304)
6.3	多处理机的软件.....	(270)	9.2.1	数据流计算机指令的组成.....	(304)
6.3.1	算术表达式的并行算法.....	(270)	9.2.2	数据流计算机指令的执行.....	(305)
6.3.2	程序并行性分析.....	(271)	9.3	数据流计算机结构.....	(306)
6.3.3	并行程序语言.....	(272)	9.3.1	静态数据流计算机模型 及其结构.....	(306)
6.3.4	多处理机的操作系统.....	(275)	9.3.2	动态数据流计算机模型 及其结构.....	(307)
6.4	多处理机系统实例.....	(278)	9.3.3	静态与动态两种数据流 计算机的比较.....	(308)
6.4.1	C _m [*] 多处理机.....	(278)	9.4	数据流程图和数据流语言.....	(309)
6.4.2	C _{mmp} 多处理机.....	(280)	9.4.1	数据流程图.....	(309)
第 7 章	RISC 结构.....	(283)	9.4.2	数据流语言及其性质.....	(313)
7.1	RISC 结构概述.....	(283)	9.5	数据流计算机的评价.....	(316)
7.1.1	传统计算机系统结构的 设计思想.....	(283)	9.5.1	数据流计算机的优缺点.....	(316)
7.1.2	RISC 设计思想的产生.....	(283)	9.5.2	数据流计算机需解决的 问题.....	(316)
7.1.3	RISC 系统结构的特点.....	(285)	第 10 章	软件对系统结构的影响.....	(318)
7.1.4	RISC 的定义.....	(286)	10.1	操作系统的影响.....	(318)
7.1.5	关于 CPI 的讨论.....	(287)	10.1.1	批量处理系统.....	(319)
7.2	流水线结构.....	(287)	10.1.2	单用户交互式系统.....	(319)
7.3	指令调度.....	(289)	10.1.3	分时操作系统.....	(319)
7.4	Cache 结构.....	(291)	10.1.4	实时操作系统.....	(320)
7.4.1	实地址 Cache.....	(292)	10.1.5	网络操作系统.....	(320)
7.4.2	虚地址 Cache.....	(294)	10.1.6	分布式操作系统.....	(320)
7.4.3	多处理器的 Cache 一致性 问题.....	(295)			

10.2	语言发展的影响	(321)	11.1.5	集群系统的并行程序设计 环境	(339)
10.2.1	实现新层次的方法	(321)	11.2	高性能计算机系统实例	(339)
10.2.2	多层计算机的设计策略 ..	(322)	11.2.1	自强 2000 的体系结构 ...	(339)
10.2.3	程序移植	(323)	11.2.2	自强 2000 的软件环境 及其特点	(340)
10.2.4	现代模型的研究方法—— 计算机仿真	(325)	11.2.3	高性能计算应用举例	(341)
10.3	并行处理的影响	(327)	11.2.4	“天河一号”的诞生	(341)
10.3.1	并行计算机分类	(327)	11.3	网格技术	(342)
10.3.2	并行计算机性能	(327)	11.3.1	网格技术的基本概念	(342)
10.3.3	并行处理技术中的基本 问题	(327)	11.3.2	网格技术简介	(343)
10.3.4	并行算法的效率与并行 机系统结构的关系	(329)	11.3.3	网格技术应用举例	(344)
10.4	面向对象程序设计的影响	(330)	11.4	云计算	(345)
10.4.1	面向对象的程序设计	(331)	11.4.1	云计算的定义	(346)
10.4.2	基于面向对象程序设计语 言的计算机系统结构	(332)	11.4.2	云计算的一个典型例子 ..	(346)
10.5	软件的固化与硬化	(334)	11.4.3	云计算的特点	(346)
第 11 章	现代计算机系统结构的发展	(336)	11.4.4	云计算的一种实现举例 ..	(347)
11.1	集群计算机系统结构	(336)	11.5	性能评价和测量	(348)
11.1.1	集群计算机系统及其 特点	(336)	11.5.1	性能评价的标志	(348)
11.1.2	集群系统的通信软件和 网络服务	(338)	11.5.2	性能的描述	(349)
11.1.3	集群系统的资源管理和 调度	(338)	11.5.3	性能评价的对象	(349)
11.1.4	集群系统的单一系统 映像	(338)	11.5.4	性能评价的手段	(349)
			11.5.5	性能的评价	(350)
			11.5.6	性能评测标准举例	(350)
			参考文献		(351)

第 1 章 计算机系统结构导论

计算机系统结构又称计算机体系结构。本章重点讨论计算机系统的功能、结构及设计方法，兼叙现代计算机系统结构的研究方向。

1.1 计算机系统的基本概念

计算机系统由紧密相关的硬件和软件组成。我们怎样从整体上来认识和分析它呢？系统（System）一词来自希腊文，是由部分组成整体的意思。从技术的角度看，我们可以给出如下定义：为完成特定任务而由相关部件或要素组成的有机整体称为系统。

我们经常使用计算机系统（Computer System）这个术语，它不难理解却又容易混淆，通常我们常听到如下三种说法。

第一种说法认为，计算机系统由运算器、控制器、存储器、输入设备、输出设备五个部件组成。其中，运算器和控制器合称中央处理器（CPU），存储器又分内存（又称主存 Memory）和外存（又称辅存 Storage）两种，所以这种说法又可简化为 I/O-CPU-M/S 模式，如图 1-1 所示。

第二种说法认为，计算机系统由硬件（Hardware）和软件（Software）两部分组成。由于技术飞速发展，昨天的软件（如现代操作系统的许多关键模块都已固化）今天已经变成了硬件，今天的硬件明天又会演变为软件，确实是硬中有软，软中有硬，相互转换，彼此渗透，这就是当今软、硬件结合的现实。事实上，对于软、硬件功能的分配及其界面的确定正是计算机系统结构（Computer Architecture）研究的内容之一，第二种说法如图 1-2 所示。

第三种说法认为，计算机系统由人员（People）、数据（Data）、设备（Equipment）、程序（Program）、规程（Procedure）五部分组成，只有把它们有机地结合在一起才能完成各种任务，我们称它为计算机系统的广义说法，如图 1-3 所示。

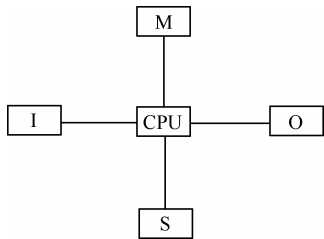


图 1-1 计算机系统说法之一

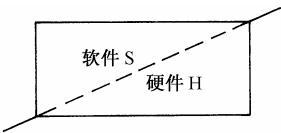


图 1-2 计算机系统说法之二

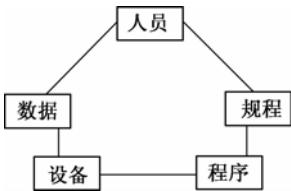


图 1-3 计算机系统说法之三

1.2 计算机系统的发展

计算机系统是经过一系列历史演变过程的产物，社会需要是一切发明与发展之母，一旦出现某种需要，人们就会奇迹般地去发明创造，去努力发展某项技术。因此，计算机系统的

发展历史不仅是一系列计算机各类系统的发展史，也是人类创造的宝贵精神财富的结晶。计算机系统结构的发展，从最初的算盘、机械式计算装置、机电式解算装置、图灵机、冯·诺依曼结构及其 EDVAC，以及到近代运用各种不同元器件所组成的各种类型的计算机系统，都是为了适应社会技术发展的进程，反过来又促进了社会与科学技术的发展。

1.2.1 冯·诺依曼体系结构的特点

存储程序概念最早由匈牙利籍数学家冯·诺依曼（Von Neumann）于 1946 年提出，他同时提出了一个完整的现代计算机雏型。60 多年来，虽然计算机经历了重大变化，性能有了惊人提高，但就其系统结构而言，进展不大，至今占有主流地位的仍是以存储程序原理为基础的冯·诺依曼结构。它由运算器（ALU）、控制器（CU）、存储器（MEM）和输入/输出设备（IN/OUT）组成，如图 1-4 所示。冯·诺依曼型计算机系统结构的基本特点可归纳如下：

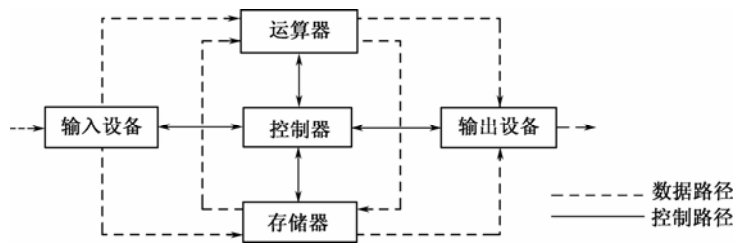


图 1-4 冯·诺依曼型计算机系统结构

（1）存储器是按地址访问的顺序线性编址的一维结构，每个单元的位数是固定的，目前均为 8 位，称为 1 字节。机器的运算速度与访问存储器的次数有关。

（2）指令由操作码和地址码组成。操作码确定本指令的操作类型和操作数的数据类型，操作数本身判定不了它是何种数据类型（如定点数或浮点数等）。地址码指明操作数的地址。

（3）指令在存储器中是按其执行顺序存储，由程序计数器指明每条指令所在单元的地址。一条指令由 1 字节至若干字节的信息组成，每取 1 字节信息，程序计数器加 1，指令取全，程序计数器指向下一条指令的地址。虽然执行顺序可以根据指令执行的结果予以改变，但从整体上来看，程序算法仍然是且也只能是顺序型的。

（4）在存储器中指令和数据同等对待。由于它们本身是无法区别的，所以指令同数据一样可以运算，即由指令组成的程序是可以修改的。

（5）计算机的系统结构以运算器、控制器为中心，输入/输出设备和存储器的数据传送都途经运算器、控制器；运算器、存储器、输入/输出设备的操作以及它们之间的联系都由控制器集中控制。

（6）指令、数据均以二进制编码表示，采用二进制运算。

冯·诺依曼等人提出的这种结构的巨大贡献在于，它奠定了计算机发展的基础。虽然当时的计算机是为解非线性微分方程而设计的，但采用这种基本结构的计算机，却成功地应用到了其他各种数值计算以及信息处理系统、事务数据处理和工业控制等广泛领域，这是冯·诺依曼等人当时所没有想到的。由于当初设计时既没有考虑要采用高级语言，也没有顾及会有操作系统以及广泛的应用领域的各种要求，由此而引起的矛盾在目前显得日益突出了。这些矛盾表明了冯·诺依曼型结构的局限性，主要有以下五点：

(1) 由于冯·诺依曼型结构以数值计算为主,因而对自然语言、图像、图形和符号处理的能力较差,不能满足今后在上述领域的应用需求。

(2) 由于程序算法从整体上是顺序型的,从而限制了并行操作的发挥,使计算机运算速度的提高因冯·诺依曼型结构的固有弱点而不能在现有基础上取得根本性的突破。

(3) 在该结构上发展起来的软件系统越来越复杂,正确性无法保证,软件生产率低下。

(4) 该结构的硬件投资较大,可靠性差,在体系结构发展上受限制。

(5) 使用该结构的计算机应用人员需要既懂专业知识,又要具备编程技巧。

传统的冯·诺依曼体系结构,为计算机的发展铺平了道路,但“集中的顺序控制”又常常成为计算机性能进一步提高的瓶颈。因此,计算机科学技术工作者仍在不断地探索和努力,寻求解决这种瓶颈的途径。

1.2.2 器件发展对系统结构的影响

计算机硬件使用的器件从电子管、晶体管、小规模集成电路迅速发展到大规模、超大规模集成电路。从电子管到小规模集成电路经历了 19 年,而到大规模集成电路 (LSI) 只用了 8 年。目前的超大规模集成电路 (VLSI) 更是以 18 月翻一番的速度发展着。器件的速度、集成度、体积、可靠性、价格等随时间以指数型模式得到改进,对提高计算机的性能价格比影响显著。器件的发展,过去、现在和将来都将是推动计算机系统结构发展的主要因素之一,而系统结构的发展也对器件提出新的更高的要求,促进其迅速发展,两者是相辅相成的。

1. 器件的功能和使用方法对系统结构的影响

按器件的功能和使用方法可将芯片分为非用户片、现场片和用户片。

非用户片的功能由制造厂商在生产时定制,使用者(即机器系统设计者)不能改变其内部功能,只能使用。这类芯片适用于各种结构,也称通用片,它有逻辑类和存储类两类。逻辑类芯片指门、触发器、多路开关、寄存器、计数器、加法器等。此类芯片要提高其集成度,势必增加若干芯片引脚。存储器类芯片容量增加一倍,仅需增加地址线引脚一根,易制造。因此,在 20 世纪六七十年代初,系统结构设计有意识地发展使用存储逻辑,尽量采用存储器件代替逻辑器件的功能。例如,用微程序控制器取代组合逻辑控制器,用 ROM 实现乘法运算、码制转换等操作(输入为地址,ROM 读出内容就是结果,类似于查表),体现“硬件软化”的某些思想。

现场片指用户可在机器组装的现场设置(或更改)其内容的芯片,如 PROM、EPROM、FPLA(现场可编程逻辑阵列)等。此类芯片灵活性好,替代硬联组合网络效率更高,加上寄存器和反馈逻辑可构成时序部件。由于现场片都是存储型芯片,规整通用,适合采用高集成技术。

完全按照用户要求设计的芯片称为用户片。对用户而言,此类芯片是全优化的。但其设计周期长、费用高、通用性差、成本大。为此,出现了半用户片 PAL 和 GAL,芯片厂商是大规模批量生产,可使成本大幅下降。而用户可根据自己需求,通过软、硬件结合的编程器对其进行“烧结”(即可编程),最终成为用户所需的芯片。

通常,同一系列的各档机器可以分别用通用片、现场片、用户片实现。就性能价格比而言,用全用户片实现的最优,用半用户片的次之,用通用片的最差。然而,从芯片设计难度、设计费用、设计周期来看,其次序正好相反。但是有一个趋势必须明确:今后,不用用户片

（包含半用户片），高速、高性能计算机是做不出来的，新型计算机系统结构也无法实现。

VLSI 的发展，产生了微处理器（MPU），将计算机系统结构、组成、实现融为一体。位片式 MPU 属于通用现场片，用户可使用软件配以不同微程序，构成用户所需指令，也可用数目不等的同一种位片式 MPU 组成不同字长的机器，从而形成功能和指令系统不同的各种系统结构的计算机。单片型 MPU 发展迅速，如 Intel 公司的 80x86 及 Pentium 系列，就属于通用型芯片，用户不能增、删其指令系统，且系列化 MPU 具有标准化趋势，在系统结构设计中占有越来越重要的地位。高性能计算机系统结构已普遍使用以 MPU 为基础的 MPP（大规模并行处理）方式或者 SMP（机群系统处理）方式。

VLSI 的发展在很大程度上取决于集成电路逻辑技术的发展，即芯片上集成的晶体管数的增长。根据近 20 年的统计，晶体管的密度（单位面积 1mm^2 上集成的晶体管数）以每年大约 35% 的速度增长，近 4 年翻两番。芯片内硅片面积每年增加 10%~20%。二者总的效果是芯片上集成的晶体管数目以每年大约 55% 的速度增长，这是半导体集成电路加工工艺发展的结果。集成电路的加工工艺是以特征尺寸来表示的，特征尺寸指在硅片上制造晶体管或晶体管之间的连线的光刻线的线宽。从 1971 年到 2001 年光刻线的线宽从 $10\mu\text{m}$ 降到了 $0.18\mu\text{m}$ 。2005 年，Intel 公司实现了 $0.065\mu\text{m}$ （即 65nm）工艺。预计，2010 年将实现 32nm 的工艺。晶体管的密度以线宽减小速度的平方增加，晶体管性能随着线宽的减小线性增加，所以晶体管数目增加的速度是性能线性增长的平方。这对计算机系统结构设计和发展来说既是挑战也是机会。

尽管线宽减小而使晶体管性能增加，但集成电路中的连线却并非如此，特别是连线上的信号延迟时间，与连线的电阻和电容的乘积成正比。当线宽减小时，连线会变短，但单位长度上的电阻和电容却会增加，因此连线延迟的改进与晶体管性能的改进相差甚远。连线延迟已经成为 VLSI 的主要设计难题，而且往往比晶体管开关延迟更为关键。越来越多的时钟周期消耗在信号传输的连线延迟上。Pentium IV 内 20 级流水线中专门分配了两级流水线在芯片内传输信号。

当集成度达到一定规模时，功耗问题凸现出来。对于现代 CMOS MPU，功耗主要在晶体管状态转换上。每个晶体管功耗与晶体管负载电容、转换（即开关）频率、电压平方的乘积成比例。当加工工艺提升时，晶体管转换的数量（即参与工作的晶体管数量）和转换频率的增长幅度压倒了负载电容和电压的减小，导致整体功耗增加。第一个 MPU 功耗仅几百 mW，而 2GHz 的 Pentium IV 功耗将近 100W。集成电路分配功耗、散发芯片热量、降低温度已经成为越来越大的难题，MPU 发展的最主要问题很可能是功耗而不是提高集成度。

2. 器件的发展对系统结构的影响

由于芯片的集成度迅速提高，其速度也在飞速提升，从而使机器的主频提高很快。最初计算机主频周期量纲是 ms (10^{-3}s) 级，1976 年出现 LSI 机器，主频周期以 μs (10^{-6}s) 计，20 世纪 80 年代初出现了 VLSI 机器，至现在主频周期以 ns (10^{-9}s) 计，不久的将来，机器主频周期将以 ps (10^{-12}s) 计。在机器组成和实现中，让处理级数最多的环节尽可能用最少的 VLSI 芯片构成，甚至采用用户片，这是缩短主频周期的重要途径。

提高主频和改进系统结构是提高机器速度的两大因素，其中系统结构改进产生的速度效应明显高于机器主频的提高，但是不能否认器件速度提高的重要性和必要性。因为从根本上讲，系统结构和组成技术的新发展、新思想能否用上，基础还在于器件的发展能否提供这种可能。如果没有器件可靠性有数量级的提高，就无法采用流水技术。如果没有高速、价廉的

半导体存储器芯片, Cache (高速缓冲存储器) 和早在 20 世纪 60 年代提出的虚拟存储器就无法真正实现。没有 PROM (EPROM) 芯片的出现, 早在 20 世纪 50 年代初就已提出的微程序控制技术也就无法真正得到广泛使用。因为有了高速相联存储器芯片, 才有相联处理机, 才能推动向量机、数组机、数据库机器的发展。

由于器件的性能价格比迅速提高, 促使新研制的组成技术加速下移。例如 Cache, 原先只有大型机才有, 20 世纪 80 年代初, 中小型机都有了, 而且容量比大型机还要大。至 20 世纪 90 年代, Cache 已出现在 PC 机内。目前芯片厂商已将 Cache 集成到 MPU 内, 如 Pentium 系列 MPU, 就有片内 Cache 和 MMU (存储器管理部件)。器件发展也使系统结构“下移”速度加快。大型机的数据表示、指令系统和操作系统出现在小型机、微型机上。MPU 使多处理机的分布处理有了扎实的基础。器件的发展还影响到算法、语言 and 软件的发展, 由多个 MPU 可以构成并行处理系统, 为了充分发挥它的高速运算能力, 必须研究并行算法、并行语言、并行处理的操作系统和应用软件。

计算机业界正在经历几十年来最重大的转型, 进入了一个崭新的时代, 性能和能效对于市场以及与计算有关的各个领域来说都极其重要, 其解决方案从晶体管级迈向了芯片级和平台级。新型操作系统, 如微软的视窗 Vista 和 Windows 7, 更为逼真的游戏、在线视频以及高清晰电视的出现对计算机的处理能力提出了新的、更高的要求。又如, 在转向高清晰度电视时, 仅用于解码的处理能力就是原来的 8 倍。在增强处理能力的同时, 减少热量、降低能耗、延长电池寿命也变得越来越重要。而发展半导体技术(硅技术)是总体解决方案的核心。Intel 新型 Core 微结构和 Intel Core 2 Duo MPU 性能基准测试处于当前世界领先水平, 而于 2007 年 11 月 Intel 推出 Intel Core 2 Extreme 四核 MPU, 其性能比双核 MPU 高出 70%。由于制造工艺的发展仍将遵循摩尔定律, Intel 计划每两年推出一种新的微结构, 到 2010 年“每瓦性能(Performance-per-watt)”将比目前产品提高 300%。2006 年 10 月, Intel 公司公布了在 300mm^2 芯片上集成 80 个 CPU 内核的 MPU (即集成 80 个浮点核心), 其运算性能高达 1TFLOPS, 相当于 1996 年 Intel 公司推出的超级计算机“ASCI 红色”系统, 该系统当时使用了近 10 000 个 Pentium Pro MPU, 安装在 85 个大型机柜内, 占地 2000 平方英尺。新型 MPU 结构采用了内存与各 CPU 内核一对一连接, 并分别拥有 256MB/s 的内存带宽和前所未有的三维架构, 即每个 CPU 内核均配备开关电路, 类似于把许多 MPU 连接成网状的超级计算架构, 在 3.1GHz 工作频率下, 平均 1W 的能耗的运算性能为 1GFLOPS, 高于现有任何 MPU 的能耗效率。芯片整体的内存带宽达到了惊人的 1TB/s。另外, Intel 公司还开发了借助于硅光子技术的光互连技术, 其传输速率达到 1Tb/s。这样, Intel 公司就拥有了以 1TFLOPS 速度工作的 MPU、1TB/s 的内存带宽和 1Tb/s 的传输速率的互连技术, 为构建全新的体系结构的计算机系统奠定了坚实的基础。

1.2.3 应用对系统结构的影响

应用对系统结构的发展有重要影响。不同的应用会对计算机系统结构的设计提出不同的要求, 其中有些要求是共同的, 如高性能价格比、友好界面、高可靠性、检测维护方便、程序可移植性和兼容性等。但是, 不同应用领域也有其特殊要求。因此, 计算机系统结构的演变与应用领域拓宽和发展有密切关系。

20 世纪 50 年代初的冯·诺依曼型计算机原本是为计算弹道、解偏微分方程而设计的,

但也适合计算其他题目，满足了当时的应用需求。20 世纪 50 年代末，其应用从科学计算扩大到商业领域的事务处理。此类应用计算量不大而输入、输出量大，要求有十进制（BCD）运算和字符行处理能力。同期，计算机应用也扩大到武器控制和工业生产过程控制等领域，从而要求计算机有较强的实时处理、A/D 和 D/A 转换、采样处理等能力，有完善的 I/O 接口，有完整的中断系统和很强的中断处理能力。由于当时的器件是电子管和晶体管，计算机体积大、成本高、可靠性低，因此，只能按不同应用需求设计成相应的专用机结构。

用户总希望机器的应用范围越宽越好，能同时支持科学计算、事务处理和实时控制。由于器件的发展，体积缩小，可靠性提高，硬件价格大幅下降，计算机制造厂商使用中小规模集成电路，因此，有可能将适合不同应用领域的专用机结构集合在一起。20 世纪 60 年代中期，以 IBM 360 为代表的具有三方面结构特点的多功能通用机相继面世。同时，出现了系列机概念，有速度不等的多档机器，用户根据应用需求进行选购和配置。这类机器又配上分时和实时操作系统，使同一型号计算机可以适应不同应用场合，这标志着计算机工业走向成熟，进入良性循环。多功能通用机概念源于大、中型机，后来下移至小、微型机，至 20 世纪 80 年代初，形成巨、大、中、小、微型机共存的局面。从系统结构观点来看，各档（型）计算机的性能随时间下移，实质上就是在低档机上引用或照搬高档机的系统结构和组成，例如原先在巨、大型机上采用的 Cache、虚拟存储器、I/O 处理机、浮点运算协处理机、复杂寻址方式和多种数据表示等均出现在现时的 PC 机上。而巨、大型机为满足高速、高性能要求，不断研制和采用新的系统结构和组成技术，如并行处理机、多处理机、向量机、相联机、阵列机、数据流机等。为了提高可靠性，出现了多种容错技术，如磁盘镜像系统、表决系统、自动检测纠错技术等。为了适应计算机网络发展的要求，出现了多种智能化的通信专用机，如智能化集线器、路由器、交换机等。随着分布处理概念的建立和发展，出现了 Client/Server（客户机/服务器）模式。

计算机应用可以归纳为数据处理（Data Processing）、信息处理（Information Processing）、知识处理（Knowledge Processing）、智能处理（Intelligence Processing），它们是逐级提升的。数据处理（包括计算）仍是计算机的主要任务之一，而 20 世纪 80 年代以来信息处理迅猛发展，数据库管理系统及其应用已成为当代计算机应用的主要领域之一，单纯用软件实现快速查询、数据修改和信息保护时，由于软件过于复杂而不能胜任，必然使庞大的数据库管理从集中式向分布式发展，典型应用的就是 Client/Server 模式。为了提高机器的智能化程度及软件硬化，目前已在开发数据库专用机。随着知识库和专家系统的发展，计算机应用将进入知识处理领域。为此，计算机系统结构必须在高速并行处理、自然语言理解、知识获取、知识表示、知识利用、逻辑推理、智能处理等方面有新的发展和突破。

1.2.4 算法对系统结构的影响

算法是软件开发的基础，也是计算机系统结构设计中为解决某个具体结构问题而建立的一套方法，如替换算法、互连算法等。一个好的算法不仅能高质量地提供一个或一类具体应用项目软件的研制和开发基础，而且能充分利用系统结构的高性能。系统结构设计者需要了解各种应用问题。但是，算法与系统结构之间总会有不协调的地方，要解决两者之间的矛盾，一般采用下列三种解决方法。

一是系统结构设计者把一些基本算法修改成更适于处理的形式。例如，设计师把一个问题

按新的方式划分,以减小存储器工作区并减少所需通用寄存器的数目;或找到算法构造的新方法,以适合并行的系统结构。因此,设计师实际上既要研究算法又要研究系统结构,充分利用算法和系统结构两方面的优势,提出一种有效的解决问题的方法。必须注意:设计师研究的算法应是一类算法,而不是一个算法,设计的系统结构应该使这类问题都能很好地解决。

二是改进基本的系统结构。过去,已经采用了高端机器技术下移的方法来改进系统的性能,如指令先取技术、Cache、流水线技术等,都已在PC机上普遍使用。为了简化指令系统,减少指令基本周期时间,达到简化机器系统结构的目的,出现了RISC(精简指令集计算机)型计算机。系统结构设计师要了解系统的瓶颈在何处,采用哪些措施可以消除瓶颈。一个好的系统当它以峰值性能工作时,应该使各部件均处于接近饱和状态。

三是利用并行性获得高速度。VLSI芯片价格大幅下降,使得并行构造的硬件成为一种可能的方案。采用低价格的器件是获得高性能价格比的最理想方案。并行系统结构提供了一种采用低成本器件技术获得高性能的方法,但是并行处理计算机能否充分发挥其效能,还需发掘程序内的并行性,即需要建立相应的并行算法,两者是相辅相成的。

为了改进系统性能,设计师可以采用改进算法、在基本系统结构上增加辅助硬件部件、设计并行系统结构这三种方法。它们都很重要,要根据具体情况,综合应用这三种方法,以期取得最佳效果。

1.2.5 价格对系统结构的影响

若要全面评价一个计算机系统的结构,就必须考虑性能价格比。如果某台机器性能提高了20%,价格也提高了20%,则性能价格比不变;若价格提高了30%,则性能价格比下降,不会受用户欢迎;若价格只提高了10%,则性能价格比上升,用户肯定欢迎。所以,在性能和价格相差不大的系统之间,采用性能价格比这个参数能较好地反映系统之间的相对性能。如果两个系统的性能和价格相差很大,则性能价格比这个参数就没有实用意义了。由此可得到改进系统结构的两条重要途径:

(1) 性能或价格在10倍以内变化而产生比原系统好的性能价格比。

(2) 提高系统的绝对性能,而价格增加比较合理。这种方法实际上会降低性能价格比,但只要用户能支付增加的费用,并认为增加的费用为他们带来的高性能可以帮助他们解决现实问题就可以了。

器件技术对计算机的性能和价格会产生影响。性能用MIPS(即 10^6 条指令/秒)表示,价格用元/MIPS表示,则性能、价格与技术形成台阶状关系。第一个台阶是20世纪80年代中期,占主导地位的器件技术是MOS技术,其中主要是NMOS,用它构造的计算机,其单位MIPS的价格几乎是个常数。第二个台阶是双极性技术ECL,用它构造的计算机其单位MIPS也是个常数。第三个台阶是专门用于高性能计算机的特殊技术,其单位MIPS价格几乎也是一个常数。器件决定了一台计算机的基本周期时间,即时钟频率。用地址总线 and 数据总线的宽度(即位数)乘以时钟频率所得值(即Mb/s),是计算机系统最大传输速率,可粗略地估计一台计算机的处理能力。设计计算机时,如果决定采用某种器件,则系统时钟频率几乎由此种器件而定。这时,改进性能的方法只能是数据总线宽度由8位改为16位、32位或64位,并相应增加存储器容量。通过这种方法所获得的性能与总线宽度成线性关系,但价格也成线性关系。因此可以得到如下结论:当某一系统结构的性能要求超过某种器件技术的极

限值时，就应采用性能更好的器件，即上升一级台阶，这导致了单位 MIPS 价格的提高。如果性能用 MIPS 描述，则 $MIPS = (\text{指令条数/周期}) \times (\text{周期数/秒}) \times 10^{-6}$ 。其中第一项与系统结构有关，第二项与采用的器件技术有关。从系统结构角度来提高系统性能，可采用下述三种方法：

- （1）优化算法，用较少的指令完成同样的任务，即减少要执行的指令数，充分利用系统结构的高性能。
- （2）在系统结构中增加硬件辅助部件，以改进系统结构的效率。如增设 Cache，可以增加每个周期执行的指令条数。
- （3）多条指令并发执行。利用重复设置的硬件增加每个周期执行的指令条数。

1.2.6 现代计算机系统的分类和发展过程

从计算机系统结构的观点来看，计算机系统分为三大类：单处理系统、并行与多处理系统、分布式处理系统。单处理系统在计算机组成原理教材中都有非常详尽的介绍，为此，本书就近代计算机系统结构中的堆栈、阵列、向量、多处理机、RISC 和 CISC、超立方体等新型系统结构的计算机系统进行了较为深入的分析与讨论。计算机系统结构的演变如图 1-5 所示。

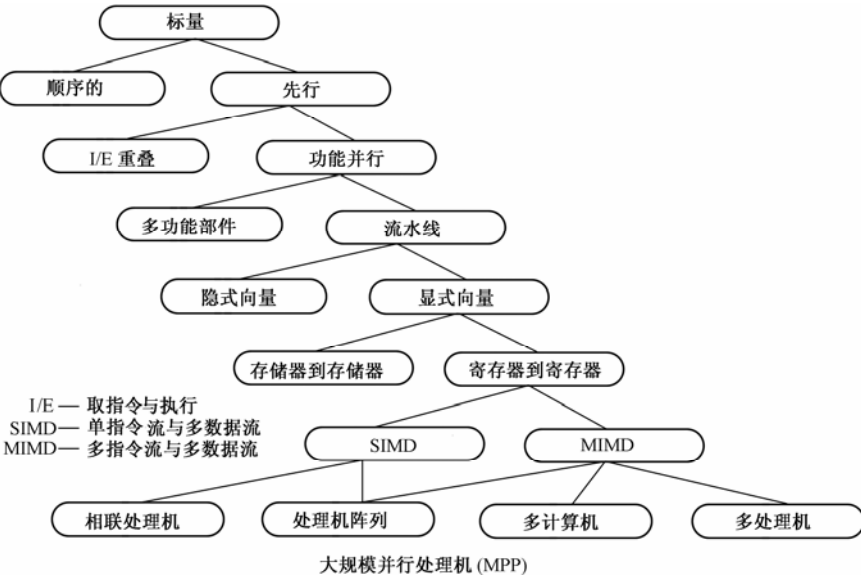


图 1-5 计算机系统结构的演变

1.3 计算机系统的功能和结构

1.3.1 计算机系统的层次结构

1. 计算机系统的功能模型

计算机系统由硬件和软件组成，二者是不可分割的整体。硬件是计算机系统实际装置，是系统的基础和核心，一般由 CPU、MEM、I/O 接口、BUS 和外部设备等组成，它以机器语言（即指令系统）提供给程序员使用。软件指操作系统、汇编程序、编译程序、文本编

辑程序、调试程序、数据库管理系统、文字处理系统、诊断程序以及各种应用程序等，基本上已与硬件的实现无关。

信息的处理过程可用控制流程的概念来描述，控制流程的实现有以下三种方法。

- (1) 全硬件方法，用组合逻辑设计方法设计硬件逻辑线路实现控制流程。
- (2) 硬件与软件相结合的方法，部分流程由微程序实现，而另一部分由硬件逻辑实现。
- (3) 全软件方法，用某种语言，按流程算法编制程序实现控制流程。

用来描述控制流程的、有一定规则的字符集合称为计算机语言。计算机使用的语言并不是专属软件范畴，它分属计算机系统各个层次，而且有不同的作用。

- 微指令是机器内部最基层的一级语言；
- 机器指令称为机器语言，是面向用户的最基层一级的语言；
- 操作系统命令从应用角度来看，也可以看作提供给用户使用的某种“语言”，用以建立一个用户应用环境；

- 符号化的机器指令（包括功能扩充的宏汇编）称为汇编语言；
- 再上一级就是用户通用的高级语言；
- 各种应用领域还有适合自己专用的语言。

用户可以在不同层次上用不同语言描述信息处理过程，但是各种语言必须翻译或解释成机器指令才能执行，所以编译或解释程序是计算机系统不可分割的一部分。基于对语言广义的理解，可以把计算机系统看成是由多级虚拟计算机组成的。从内向外，层层相套，形成“洋葱”式结构的功能模型，如图 1-6 所示。

该模型的每一层次都是一个虚拟计算机，其组成如图 1-7 所示。所谓虚拟计算机是指只对观察者而存在的计算机，它的功能体现在广义语言上，对该语言提供解释手段，然后作用在信息处理或控制对象上，并从对象上获得必要的状态信息。从某一层次的观察者看来，他只能通过该层次的语言来了解和使用计算机，至于内层如何工作和实现功能他就不必关心了。简而言之，虚拟计算机就是由软件实现的机器。

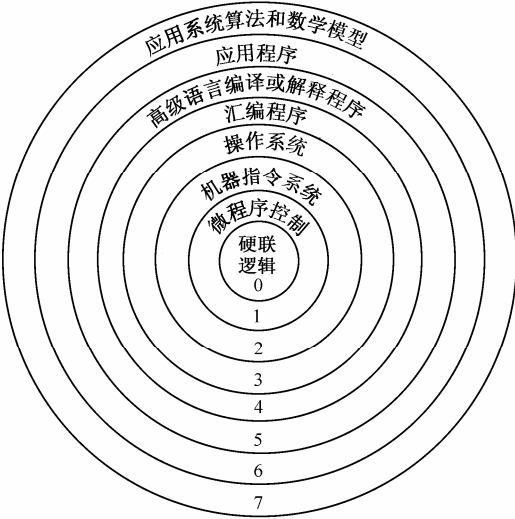


图 1-6 计算机系统功能模型

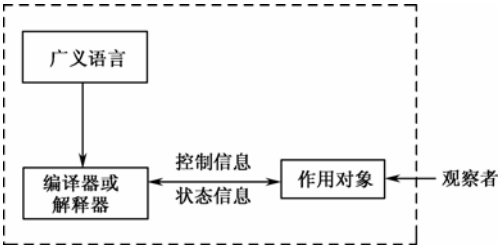


图 1-7 虚拟计算机的组成

2. 计算机系统的功能层次

用上述虚拟计算机的观点来定义计算机系统，就得到图 1-8 所示的功能层次，并在各层次注有观察者身份。

- M0 级为硬联逻辑，是实现微指令本身的控制时序。
- M1 级为微程序控制，是对机器指令进行译码，对应一个微指令序列，给出微操作信号。
- M0 级和 M1 级实现了中央处理机功能。

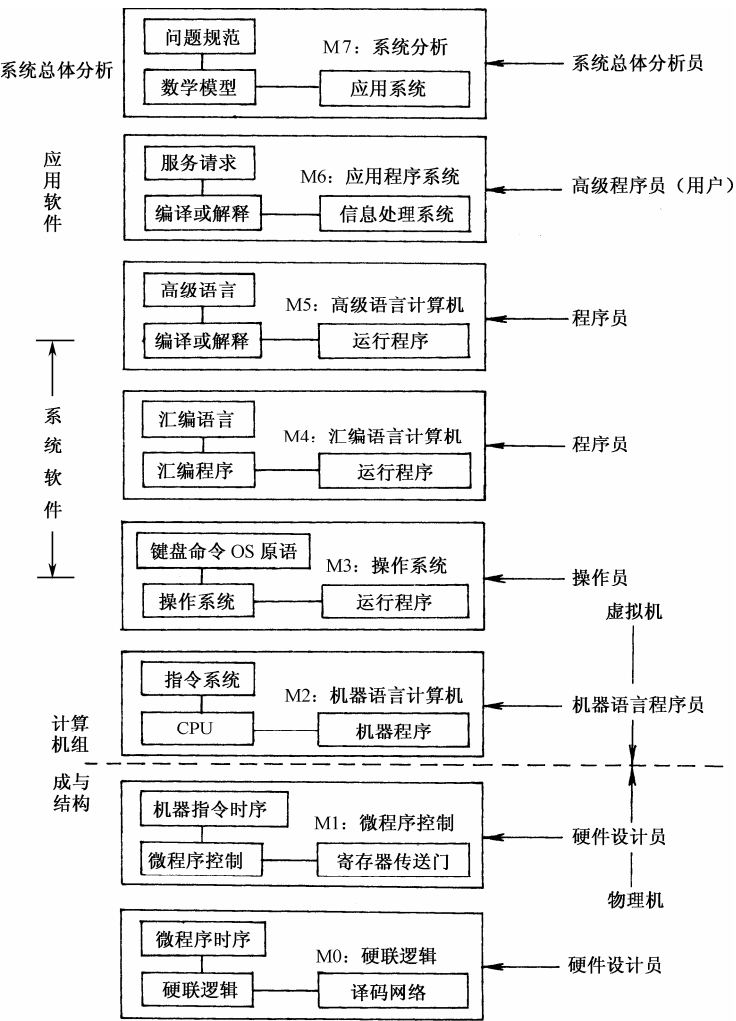


图 1-8 用虚拟计算机观点定义计算机系统的功能层次

- M2 级为机器语言计算机，面向用户的计算机指令系统。
 - M3 级为操作系统，为用户提供一个操作环境，提高了计算机系统的功能和资源利用效率。
 - M4, M5 级为语言类计算机，供程序员使用，基本上脱离了物理计算机。
 - M6 级为应用程序系统，是供非专业人员使用的计算机。
 - M7 级为系统分析，建立数学模型和算法，确定系统配置。
- 从学科领域划分看，M0~M2 级是计算组织与结构的范围；M3~M5 级是系统软件的范

围；M6级是应用程序的范围；M7级是系统总体分析的范围。当然级与级之间存在某些交叉。例如，M2级涉及汇编语言程序设计；M3级处于硬件向软件过渡；M6级处于系统软件向应用系统过渡。此外，在特殊计算机系统中，有些级别可能不存在。

总之，我们强调把计算机系统当作一个整体，既包含硬件，也包含软件，软件和硬件在逻辑功能上是等效的，即某些操作由软件（或由硬件）实现，反之亦然。故软、硬件之间没有固定不变的分界面，主要受实际应用需要及系统性能价格比所支配，图1-8所示M1级为物理机（硬件）与虚拟机（软件）界面，它仅对一般情况而言，从用户来看，机器的速度、可靠性、可维护性是主要的硬件技术指标。具有相同功能的计算机系统，其软、硬件之间的功能分配可以有很大差异，随着组成计算机的基本元器件的发展，其性能不断提高，价格不断下降，因此硬件成本下降。与此同时，随着应用不断发展，软件成本在计算机系统中所占比例上升。这就造成了软、硬之间界面推移，某些软件完成的工作由硬件去完成（即软件硬化），同时也提高了计算机实际运行速度。

1.3.2 计算机系统结构定义

计算机系统结构（Computer Architecture）也称计算机体系结构。1964年G. M. Amdahl在介绍IBM 360系列机时提出，20世纪70年代广泛采用。由于器件技术迅速发展，计算机硬、软件界面在动态变化，因此对计算机系统结构定义的理解也不尽一致。

Amdahl提出：计算机系统结构是从程序设计者所看到的计算机的属性，即概念性结构和功能特性，这实际上是计算机系统的外特性。然而从计算机系统的层次结构概念出发，不同级的程序设计者所看到的计算机的属性显然是不一样的，见图1-8。因此，所谓“系统结构”是指计算机系统中对各级之间界面的定义及其上、下级的功能分配。所以，各级都有其自己的系统结构。各级之间存在“透明性”。所谓“透明性”，一是指确实存在，二是指无法监测和设置。在计算机系统中，低层的概念性结构和功能特性，对高层来说是“透明”的。本课程讲述的计算机系统结构是指图1-8中的M2级——机器语言级计算机。其界面之上是所有软件功能，界面之下是所有硬件和固件功能。因此，这个界面实际上是软件和硬件的分界面。所以，计算机系统结构是指对机器语言计算机的软、硬件功能分配和对界面的定义。

计算机系统机构的研究对象是计算物理系统的抽象和定义，具体包括：

数据表示——定点数、浮点数编码方式，硬件能直接识别和处理的数据类型和格式等；

寻址方式——最小寻址单位，寻址方式种类，地址计算等；

寄存器定义——通用寄存器、专用寄存器等定义、结构、数量和作用等；

指令系统——指令的操作类型和格式，指令间排序和控制（微指令）等；

存储结构——最小编址单位、编址方式、主存和辅存容量、最大编址空间等；

中断系统——中断种类，中断优先级和中断屏蔽，中断响应，中断向量等；

机器工作状态定义和切换——管态、目态等定义及切换；

I/O系统——I/O接口访问方式，I/O数据源、目的、传送量，I/O通信方式，I/O操作结束和出错处理等；

总线结构——总线通信方式、总线仲裁方式、总线标准等；

系统安全与保密——检错和纠错，可靠性分析，信息保护，系统安全管理等。

1.3.3 计算机组成与实现

计算机组成（Computer Organization）指计算机系统结构的逻辑实现，包括机器级内的数据通道和控制信号的组成及逻辑设计，它着眼于机器级内各时间的时序方式与控制机构、各部件功能及相互联系。

计算机组成还应包括：数据通路宽度；根据速度、造价、使用状况设置专用部件，如是否设置乘法器、除法器、浮点运算协处理器、I/O 处理器等；部件共享和并行执行；控制器结构（组合逻辑、PLA、微程序）、单处理机或多处理机、指令先取技术和预估、预判技术应用等组成方式的选择；可靠性技术；芯片的集成度和速度的选择。

计算机实现（Computer Implementation）指计算机组成的物理实现，包括处理机、主存等部件的物理结构，芯片的集成度和速度，芯片、模块、插件、底板的划分与连接，专用芯片的设计，微组装技术，总线驱动，电源、通风降温、整机装配技术等，它着眼于芯片技术和组装技术，其中，芯片技术起着主导作用。

1.3.4 计算机系统结构、组成和实现三者的关系

计算机系统结构、组成和实现是三个不同的概念。系统结构是计算机系统的软、硬件界面；计算机组成是计算机系统结构的逻辑实现；计算机实现是计算机组成的物理实现。它们各自有不同的内容，但又有紧密的关系。

例如，指令系统功能的确定属于系统结构，而指令的实现，如取指、取操作数、运算、送结果等具体操作及其时序属于组成，而实现这些指令功能的具体电路、器件设计及装配技术等属于实现。

又如，是否需要乘、除指令属于系统结构，而乘、除指令是用专门的乘法器、除法器实现，还是用加法器累加配上右移或左移操作属于实现，而乘法器、除法器或加法器的物理实现，如器件选择及所用微组装技术等属于实现。

再如，对主存、主存容量与编址方式（即按位、按字节还是按字访问）的确定属于系统结构，而主存的速度、逻辑结构、性能价格比等属于实现，至于存储器芯片选定、片选译码电路设计、主存部件组装连接等则属于实现。

由此可见，具有相同系统结构（如指令系统相同）的计算机可以因为速度要求不同等因素而采用不同组成。例如，取指、译码、取数、运算、存结果可以顺序进行，或采用时间上重叠的流水线技术以提高执行速度。又如，乘法指令可以采用专门的乘法器，或采用加法器通过累加、右移实现，这取决于机器要求的速度、程序中乘法指令出现频度及所采用的乘法算法。如出现频度高、速度快可用乘法器；出现频度低，用后一种方法对机器整体速度下降影响不大，却可显著降低价格。

同样，一种计算机组成可以采用多种不同的计算机实现。例如，主存可用 TTL 芯片或 MOS 芯片，也可用 LSI 工艺芯片或 VLSI 工艺芯片，这取决于器件技术和性能价格比。

总而言之，系统结构、组成和实现之间的关系应符合下列原则：系统结构设计不要对组成、实现有过多和不合理限制；组成设计应在系统结构指导下，以目前能实现的技术为基础；实现应在组成的逻辑结构指导下，以目前器件技术为基础，以性能价格比优化为目标。

1.3.5 计算机系统的特性

计算机系统从功能到结构都具有明显的多层次性质，此外，从不同角度看计算机系统，还具有许多其他重要特性。

1. 计算机等级

通常把计算机系统按其性能与价格的综合指标分为巨型、大型、中型、小型、微型等若干级。但是，随着技术的进步，各级计算机的性能指标都在不断提高，以至于30年前的一台大型机的性能甚至比不上当今一台微型计算机。可见，划分计算机等级的绝对性能标准是随时间变化的。假定以不变的绝对价格标准来划分计算机等级，便可得到如图1-9所示的计算机等级与价格、性能的关系示意曲线。

计算机等级的发展遵循以下三种不同的设计思想：

(1) 在本等级范围内以合理的价格获得尽可能高的性能，逐渐向高档机发展，称为最佳性能价格比设计。

(2) 只求保持一定的合用的性能而争取最低价格，称为最低价格设计，其结果往往是从低档向下分化出新的计算机等级。

(3) 以获取最高性能为主要目标而不惜增加价格，称为最高性能设计，由此产生当前最高等级的计算机。

第(1)类设计主要针对大、中型计算机用户的需要，生产主计算机以及超级小型机；第(2)类设计以普及应用计算机为目标，生产数量众多的微、小型计算机；第(3)类设计只满足少数用户的特殊需要，在数量上不占主流。图1-9中斜虚线代表等性能线，它反映了较高等级计算机技术措施向较低等级计算机推广及转移的趋势。

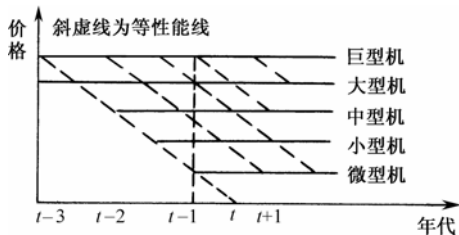


图1-9 计算机等级与价格、性能关系示意图

2. 计算机系列

系列机概念指先设计好一种系统结构，而后就按这种系统结构设计它的系统软件，按器件状况和硬件技术研究这种结构的各种实现方法，并按照速度、价格等不同要求，分别提供不同速度、不同配置的各档机器。系列机必须保证用户看到的机器属性一致。如IBM AS400系列机，其数据总线有16, 32, 64位之分，而数据表示方式一致。系统软件必须兼容，系列机软件兼容指同一个软件（目标程序）可以不加修改地运行于系统结构相同的各个机器上，而且所得结果一致。软件兼容有向上兼容和向下兼容两个含义：向上兼容指低档机器的目标程序（机器语言级）不加修改就可以运行于高档机器；向下兼容指高档机器的目标程序不加修改可以运行于低档机器。一般不使用向下兼容方式，软件的前后兼容指按系列机投放市场先后，实现软件兼容。一般是向后兼容。

计算机系列化有以下7个方面的优点：

- (1) 在使用共同系统软件的基础上解决程序兼容性问题；
- (2) 在统一数据结构和指令系统的基础上，便于组成多机系统和网络；
- (3) 使用标准的总线规程，实现接插件和扩展功能卡兼容，便于实现OEM（由各厂生产功能卡，然后组装成系统）；
- (4) 扩大计算机应用领域，提供用户在同系列的多种机型内选用最合适机器的可能性；

- (5) 有利于机器的使用、维护和人员培训；
- (6) 有利于计算机升级换代；
- (7) 有利于提高劳动生产率，增加产量，降低成本，促进计算机的发展。

3. 模拟与仿真

系列机能实现程序移植，其原因在于系列机有相同的系统结构。如果要求程序能在具有不同系统结构的机器间相互移植，就要做到在某一系统结构之上实现另一种系统结构，即实现另一种机器的属性，从指令系统角度看，就是在这一系统结构上实现另一种指令系统，即另一种机器语言。前面已述，计算机系统可按功能划分成不同层次，如把上述一种机器语言也作为虚拟计算机语言看待，就可以用相似的层次结构来实现，如图 1-10 所示。图中要求原在 B 机器运行的程序，能移植于 A 机器，因此在 A 机器语言级上，用虚拟机的概念实现 B 机器的指令系统。B 机器的每一条机器指令由一段 A 机器语言程序去解释执行。这种用机器语言程序解释实现程序移植的方法称为模拟，B 机器称为虚拟机，A 机器称为宿主机。

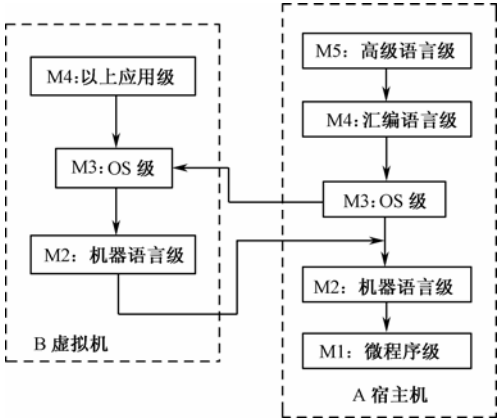


图 1-10 在 A 机器语言级上实现 B 机器指令系统

若机器采用微程序控制，则 B 机器指令需通过二重解释，显然，如果直接用微程序去解释 B 机器指令就会加快解释过程。这种用微程序直接解释另一种机器的指令系统称为仿真。

仿真与模拟的主要区别在于解释用的语言。仿真是用微程序解释，其解释程序在微程序存储器；模拟是用机器语言程序解释，其解释程序在主存储器。

为了模拟 B 机器的全貌，除了模拟指令系统外，还要模拟 B 机器的存储系统、操作系统、I/O 系统等，这些模拟程序的编制十分复杂，由于 B 机器的一条指令不是由硬件直接执行，而是经多条 A 机器指令的软件模拟实现，故速度显著下降。所以，模拟只适合于移植运行时间不长，使用次数少的程序。仿真可提高速度，但难于仿真存储系统及 I/O 系统，且只能在系统结构差距不大的机器间运用。在开发系统中，模拟和仿真往往并用，以增强开发系统的功能。

模拟方法灵活性大，从原理上讲它可以实现程序在任何机器间的相互移植，然而其效率很低，速度损失很大。仿真方法在速度上损失要小得多，但目前只是在系统结构差别不大的机器之间采用，否则效率也会降低，而且还需和模拟方法结合才能真正实现。随着软件工程技术的发展，以及软件工具和环境的建立，越来越多的应用软件采用高级语言编制，就不一定非要将汇编语言级的兼容放在首位了。这时，操作系统的兼容就提出来了。目前，众多的微机系统使用的操作系统集中在少数几种，就反映了这个要求。

1.4 计算机系统设计的方法

系统结构设计是整个计算机系统设计的基础，因此，如何确定系统结构与计算机系统的设计方法有很大关系。

1.4.1 软、硬件取舍的基本原则

系统结构设计的主要任务是进行软、硬件功能分配和给用户提机器级软/硬界面。相同功能的计算机系统，其软、硬件功能分配的比例可以在很宽的范围内变化。一般来说，提高硬件功能比例可以提高运算速度，减小存储容量，但硬件成本上升，降低了硬件利用率和系统的灵活性与适应性；而提高软件功能比例可以降低硬件成本，提高系统的灵活性与适应性，但运算速度会下降，存储容量和软件开发费用要增加。因此，软、硬功能分配比例应考虑在现有硬件和芯片条件下，争取系统有较高性能价格比。

性能含义极为广泛，它是一些指标的综合，其中速度是性能中的重要指标之一。软、硬件功能分配可用实现费用、速度和其他性能要求来考虑。

软、硬件功能分配的第一个方面是分析实现费用。软、硬件实现费用包括研制费用 D 和重复生产费用 M 。软、硬件研制费用分别为 D_S 和 D_H ，而 $D_H \approx 100D_S$ 。软、硬件重复生产费用分别为 M_S 和 M_H ，则 $M_H \approx 100M_S$ 。这是因为硬件（主要指芯片）研制和重复生产的投入比软件大得多，芯片研制费用不仅包含芯片本身的研发费用，还应摊入生产线成本、材料成本，重复生产也是如此。而软件开发费用涉及面较小，投入的主要是智力成本，而其重复生产成本（即复制）费用微乎其微。当然，两者的比例并非一成不变，现暂取 100。

用硬件实现一个功能（如子程序调用的全部操作）往往只需设计一次，而用软件实现时，由于应用程序调用该功能时环境不同，需做修改或重新设计。设 C 为软件实现该功能时重新设计的次数，则该功能用软件实现费用为 CD_S （由于重新设计时可利用原设计进行修改，甚至简单复制，因此 D_S 可低很多）。软件在存储介质上出现了 R 次，软件实现该功能的重复生产费用为 RM_S 。若该型计算机系统共生产了 V 台，每台计算机用硬件实现该功能费用是

$$\frac{D_H}{V} + M_H$$

改用软件实现，费用是

$$\frac{CD_S}{V} + RM_S$$

只有当

$$\frac{D_H}{V} + M_H < \frac{CD_S}{V} + RM_S$$

时，用硬件实现才适宜。将 $D_H=100D_S$ 和 $M_H=100M_S$ 代入上式

$$\frac{100D_S}{V} + 100M_S < \frac{CD_S}{V} + RM_S$$

上式只有在 C 和 R 值均大于 100 时才成立。也就是说，该功能是经常用到的基本功能，才适合用硬件实现，不能认为硬件比例越大越好。

另外，软件初始设计费用远比重复生产费用高， $D_S \approx 10^4 M_S$ 是完全可能的。将此式代入上式，得

$$\frac{10^6}{V} + 100 < \frac{10^4 C}{V} + R$$

由上式可得
经整理，得

$$10^6 + 100V < 10^4 C + RV$$
$$10^4 (10^2 - C) < (R - 100) V$$

因为 C 值一般总比 100 小， R 值往往大于 100，所以 V 值愈大，不等式才成立。即只有产量 V 大的计算机系统，用硬件实现才是适宜的。

软、硬件功能分配的第二个方面是考虑组成技术，并使它不要过多或不合理地限制各种组成、实现技术的采用。

软、硬件功能分配的第三个方面是，不仅从“硬”角度充分考虑应用组成技术成果和器件技术进展，还必须从“软”角度考虑为编译、操作系统、程序设计提供好的硬件平台。早期由于器件贵、可靠性差、体积大，所以在软、硬件功能分配时倾向于软件，尽量把负担交给软件以简化硬件。而近 20 年来，芯片制造工艺飞速发展，若再维持早期观念，将阻碍计算机系统结构的发展。所以，必须考虑软件硬化，为软件提供更好的界面，为用户提供更好的程序设计环境。

1.4.2 计算机系统设计的定量原则

下面介绍计算机系统设计中经常用到的几个定量原则。

1. 加快经常性事件的速度（make the common case fast）

这是计算机设计中最重要且应用最广泛的设计原则。使经常发生的事件的处理速度加快能明显提高整个系统的性能。例如，CPU 进行加法运算时，结果可能产生溢出，但更多时间是无溢出的。因此加快无溢出时的处理速度，对溢出处理不过多优化，这样可产生最佳效益。

2. Amdahl定律

如何确定经常性事件以及如何加快处理它，是 Amdahl 定律需要解决的问题。Amdahl 定律如下：系统中某部件因采用某种更快执行方法后，整个系统性能的提高与这种执行方式使用频率或占总执行时间的比例有关。

Amdahl 定律定义了加速比：

$$\text{加速比} = \frac{\text{采用改进措施后的性能}}{\text{未采用改进措施前的性能}} = \frac{\text{未采用改进措施前执行某任务的时间}}{\text{采用改进措施后执行某任务的时间}}$$

加速比与两个因素有关：

$$F_e = \frac{\text{可改进部分占用时间}}{\text{改进前整个任务执行时间}} \quad (F_e < 1), \quad S_e = \frac{\text{改进前改进部分执行时间}}{\text{改进后改进部分执行时间}} \quad (S_e > 1)$$

由此，得到下列结论。

(1) 改进后整个任务执行时间

$$T_n = T_o \left[(1 - F_e) + \frac{F_e}{S_e} \right]$$

式中， T_o 为改进前整个任务执行时间。

(2) 改进前后整个系统的加速比

$$S_n = \frac{T_o}{T_n} = \frac{1}{(1-F_e) + \frac{F_e}{S_e}}$$

式中, $(1-F_e)$ 表示不可改进部分, 当 $F_e=0$, 即无改进部分时, $S_n=1$, 所以性能提高幅度受改进部分所占比例限制。当 $S_e \rightarrow \infty$ 时, 则 $S_n = \frac{1}{1-F_e}$, 获取的性能改善极限值受 F_e 的约束。

【例 1-1】 设某系统的某部件原处理时间占整个运行时间的 40%, 现加快 10 倍速度, 则整个系统性能提高多少?

解: 由题意可知 $F_e=0.4$, $S_e=10$, 则

$$S_n = \frac{1}{0.6 + \frac{0.4}{10}} = \frac{1}{0.64} \approx 1.56$$

【例 1-2】 设求浮点数平方根 FPSQR 的操作占整个测试程序执行时间的 20%。一种实现方法是采用 FPSQR 硬件, 使其速度加快 10 倍; 另一种实现方法是使所有浮点数指令 FP 速度加快 2 倍, 同时设 FP 指令占整个程序执行时间的 50%。请比较两种实现方法的优劣。

解: 硬件方案 $F_e=0.2$, $S_e=10$, 则

$$S_n = \frac{1}{(1-0.2) + \frac{0.2}{10}} = \frac{1}{0.82} = 1.22$$

FP 加速方案 $F_e=0.5$, $S_e=2$, 则

$$S_n = \frac{1}{(1-0.5) + \frac{0.5}{2}} = \frac{1}{0.75} = 1.33$$

可见 FP 加速方案更好。但需注意, 这是在程序量的 50% 是 FP 指令的前提下。

3. CPU性能公式

CPU 性能取决于三个要素: ①时钟频率 f ; ②每条指令时钟周期数; ③指令条数 I_C 。时钟周期 $T=1/f$ 。一个程序执行时的 CPU 时间可表示为

$$\text{CPU 时间} = \frac{\text{CPU 时钟周期总数}}{\text{时钟频率 } f}$$

或

$$\text{CPU 时间} = \text{CPU 时钟周期总数} \times \text{时钟周期 } T$$

如果一个程序的指令条数为 I_C , 则每条指令平均时钟周期数

$$\text{CPI} = \frac{\text{CPU 时钟周期总数}}{I_C}$$

经代换, 可得

$$\text{CPU 时间} = I_C \cdot \text{CPI} \cdot T = I_C \cdot \text{CPI} \cdot \frac{1}{f}$$

一个程序的 CPU 时钟周期总数也可用下列方法计算:

$$\text{CPU 时钟周期总数} = \sum_{i=1}^n \text{CPI}_i \cdot I_i$$

式中, I_i 是 i 指令在程序中的条数, CPI_i 为 i 指令的平均时钟周期数, n 为程序中指令的种类数, 此时

$$\text{CPU 时间} = \sum_{i=1}^n (\text{CPI}_i \cdot I_i) \cdot T$$

则

$$CPI = \frac{\sum_{i=1}^n CPI_i \cdot I_i}{I_C} = \sum_{i=1}^n CPI_i \cdot \frac{I_i}{I_C}$$

式中， $\frac{I_i}{I_C}$ 表示 i 指令在程序中所占比例。

【例 1-3】 如果 FP 操作比例为 25%，FP 的 CPI=4.0；其他指令的 CPI=1.33；FPSQR 操作比例为 2%，FPSQR 的 CPI=20。如有两种方案，分别把 FPSQR 操作的 CPI 和所有 FP 操作的 CPI 减至 2。试利用 CPU 性能公式分析该两个方案哪个更好。

解：原系统时钟频率 f 和指令条数 I_C 保持不变时，没有采取提高措施之前，原系统

$$CPI = \sum_{i=1}^n CPI_i \cdot \frac{I_i}{I_C} = (4 \times 25\%) + (1.33 \times 75\%) = 2.0$$

采用方案 1（使 FPSQR 操作的 CPI 为 2）后，整个系统的 CPI 为

$$CPI_1 = CPI - (CPI_{\text{原 FPSQR}} - CPI_{\text{新 FPSQR}}) \times 2\% = 2.0 - (20 - 2) \times 2\% = 1.64$$

采用方案 2（使 FP 操作的 CPI 为 2）后，整个系统的 CPI 为

$$CPI_2 = CPI - (CPI_{\text{原 FP}} - CPI_{\text{新 FP}}) \times 25\% = 2.0 - (4.0 - 2) \times 25\% = 1.5$$

也可根据下式计算 $CPI_2 = (1.33 \times 75\%) + (2.0 \times 25\%) = 1.5$

显然， $CPI_2 < CPI_1$ ，方案 2 比方案 1 好。

方案 2 加速比

$$S_2 = \frac{\text{原系统 CPU 时间}}{\text{方案 2 CPU 时间}} = \frac{I_C \cdot T \cdot CPI}{I_C \cdot T \cdot CPI_2} = \frac{CPI}{CPI_2} = \frac{2}{1.5} = 1.33$$

方案 1 加速比

$$S_1 = \frac{CPI}{CPI_1} = \frac{2}{1.64} = 1.22$$

所以， $S_2 > S_1$ 。

【例 1-4】 设有两台机器 A 和 B，对条件转移采用不同方法。CPU_A 采用比较指令和条件转移指令处理方法，若条件转移指令占总执行指令数的 20%，比较指令也占 20%。CPU_B 采用比较和条件转移指令合一方法，占总执行指令数的 20%。若规定两台机器执行条件转移指令需 $2T$ ，其他指令需 $1T$ 。CPU_B 的条件转移指令比 CPU_A 慢 25%，现 CPU_A 和 CPU_B 哪个工作速度更快？

解：

$$CPI_A = 0.2 \times 2 + 0.8 \times 1 = 1.2$$

$$CPU_A \text{ 时间} = I_{C_A} \cdot CPI_A \cdot T_A = 1.2 T_A \cdot I_{C_A}$$

式中， I_{C_A} 是 CPU_A 的指令条数，由于 CPU_B 无比较指令，因此 $I_{C_B} = 0.8 I_{C_A}$ ；若 $I_{C_A} = 100$ ，则 $I_{C_B} = 80$ ；而 CPU_B 的条件转移指令仍是 20 条，所以占比为 $\frac{20}{80} = 0.25 = 25\%$ 。

$$CPI_B = 0.25 \times 2 + 0.75 \times 1 = 1.25$$

又因为 CPU_B 的 T_B 比 CPU_A 的 T_A 慢 25%，所以 $T_B = 1.25 T_A$ 。

$$CPU_B \text{ 时间} = I_{C_B} \cdot CPI_B \cdot T_B = 0.8 I_{C_A} \times 1.25 \times 1.25 T_A = 1.25 T_A \cdot I_{C_A}$$

可见，CPU_A 时间 < CPU_B 时间，CPU_A 比 CPU_B 工作速度快。

【例 1-5】 在例 1-4 中， T_B 只比 T_A 慢 10%，问哪个 CPU 更快些？

解：

$$T_B = 1.1 T_A$$

$$\text{CPU}_B \text{ 时间} = 0.8 I_{C_A} \times 1.25 \times 1.1 T_A = 1.1 T_A \cdot I_{C_4}$$

所以 CPU_B 时间 $<$ CPU_A 时间, CPU_B 更快些。

4. 程序访问的局部性原理

经过对大量典型的程序进行统计分析可以发现, 在一个时间片内, 90% 的时间去执行 10% 的程序代码, 即大部分时间是访问程序的局部空间。程序访问的局部性原理是构建存储体系和建立 Cache 的理论基础, 详情请参阅下面有关章节。

1.4.3 计算机系统的设计任务

计算机系统设计涉及系统结构、组成和实现的方方面面。要设计出优化的方案, 还需要熟悉编译器和操作系统等方面的技术。计算机系统设计的主要任务有下列三个方面。

1. 确定用户对计算机系统的功能、价格和性能要求

计算机系统的功能是根据市场需求而定的。应用软件或应用市场往往对功能确定起决定性作用。如果某一类应用软件是基于某一指令子集的, 那么在新的系统结构设计时要包容并实现该指令子集。如果某一类应用有很大市场, 那么新的系统设计必须考虑适用于这类应用。具体功能要求包括:

(1) 应用领域。首先明确是专用还是通用? 专用机应用于特定领域, 特殊性能要求会很高。通用机适应各种应用场合, 要求有比较平衡的性能。其次是确定面向科学计算还是面向事务处理。科学计算应用领域应增强浮点运算性能, 而事务处理则要求系统支持数据库。

(2) 软件兼容层次。如要求在程序设计语言层次兼容, 只需有新的编译器即可, 对于系统结构设计而言, 最具灵活性。若要求目标代码层次兼容, 则系统结构完全确定, 灵活性差, 如系列机, 但不需要在软件或程序移植方面进行投资。

(3) 操作系统需求。主要集中于存储系统设计: 如直接寻址空间范围 (即主存容量), 某种程度上决定应用程序的大小; 又如存储管理部件 (MMU) 设置和存储保护等。

(4) 标准。有运算方面的, 如浮点数标准, 已有 IEEE, DEC, IBM 等格式和算法。有底板上系统总线标准, 如 ISA, EISA, MULTIBUS I, MULTIBUS II, VME, MCA, PCI 等。有 I/O 总线标准, 如 SCSI, IEEE488, USB, IEEE1394, RS-232C, RS-422A, IEEE1284 等。有网络标准, 如 TCP/IP, X.25, X.21, IEEE802 系列, Novell Netware, ARPANET, NSFNET 等, 适用于 Ethernet, ATM, ISDN, ADSL, DDN, FDDI, 帧中继等网络。有程序设计语言标准, 如 ANSI C, FORTRAN 77, ANSI COBOL 等。为适用不同标准, 在计算机系统设计时, 应做相应的处理。

2. 软、硬件平衡

一旦待设计机器的功能确定下来, 下一步必须考虑设计的优化。最优方案的选择取决于衡量标准的选择。通常用性能价格比来考核。应用领域确定后, 可用该领域中具有代表性的应用程序来衡量计算机性能。优化设计必须考虑软、硬件合理分配。一种功能由软件实现还是用硬件实现各有长处。用软件实现设计容易, 修改方便; 用硬件实现速度快, 有较好性能。但并非硬件实现一定比软件实现速度快, 算法在其中起重要作用, 一个先进算法用于软件实现其体现的速度可快于较差算法用于硬件实现。对于特殊需求的计算机应该配以相应的硬件以满足其性能要求, 例如用于科学计算的计算机, 就应该配以浮点协处理器, 以提高运算速度, 若以软件实现浮点运算将使速度明显下降, 严重影响整机性能。以事务处理为主的计算机系统则在指令系统中必有二-十进制数 (BCD)、字符串操作等指令。所以, 协调软、硬件

分配使之达到平衡是得到最佳性能价格比的重要途径之一。

同时，在方案选择中，设计的复杂性也是必须考虑的一个因素，复杂设计花费时间长，因此这样的设计必须有较高性能才有竞争力。一般而言，用软件实现复杂设计较容易。另一方面，指令系统和组织结构设计会影响实现的复杂性和编译器、操作系统的复杂性。因此，软、硬件平衡也应该包含软、硬件实现的难易程度。

3. 系统结构设计应符合今后发展的方向

一个成功的系统结构设计应能承受软、硬件发展和应用的变化。因此，设计时必须注意计算机技术和计算机应用的发展趋势，这样才能延长机器的使用寿命。芯片技术发展促进了软件硬化，在硬件设计中必须考虑易扩性、兼容性，便于今后机器的升级换代。应用发展使程序和数据飞速膨胀，因此存储系统（尤其是主存）必须要有可扩性。软件的发展趋势是普遍用高级语言代替汇编语言，这就要求系统结构对编译器的支持进一步提升。面向对象程序设计技术的推广，要求系统结构设计有相应的措施。

1.4.4 计算机系统的设计步骤

计算机系统从概念和功能上可看成由多级构成的层次结构（如图 1-8 所示），从哪一层开始设计是有影响的。通常有“由上往下”、“由下往上”和“由中间开始”三种不同的设计思路。

“由上往下”设计方法首先考虑如何满足用户要求，即首先确定面向用户的虚拟机器的基本特性和环境，然后再逐级往下设计，直到硬件执行那一级为止。在每级设计过程中，要考虑怎样使上一级能优化实现。这样设计出来的计算机系统对于应用效能必然是很好的。这种方法适用于某些专用机的设计，对于通用机设计往往不适用。因为“由上而下”设计出来的机器在应用对象或应用范围发生变化时，软、硬分配就会很不适应，导致系统效率急剧下降。

“由下往上”设计方法是根据当时器件，参照、吸收已有各种机器的特点，从微程序机器级（如果采用微程序控制）及传统机器级开始研制，然后再配以不同应用领域的多种操作系统和编译器，这是一种通用机的设计方法。但是由于它是在硬件已定情况下被动地设计软件的，尽管某些硬件设计对于软件实现不利，也无法加以改变。这样势必造成软、硬件脱节，软件因得不到硬件支持而无法优化。而且，研制出来的机器的某些性能指标还可能是虚假的。例如，传统机器级的“每秒运算次数”指标就是如此。如指令系统中有面向操作系统和编译的指令，则其效果往往比单纯将“运算次数”提高 10%~20% 效果还要大。这种设计方法在芯片技术飞速发展的今天，很难适应系统设计要求。

“由中间开始”的设计方法是从层次结构中的软、硬交界面开始，即从传统机器级与操作系统级之间（即机器语言级）出发，合理地进行软、硬件功能分配，既考虑芯片，又考虑应用算法和数据结构等。该交界面定义后，才分别往上、往下进行软件和硬件设计。软件和硬件的设计可分别同时进行，有利于缩短设计周期，也有利于软、硬交互协调，因此它是一种较好的设计方法。它要求设计人员有丰富的软/硬件、芯片和应用等方面的知识。为了能在硬件研制出来之前开展软件设计调试，还应选择有效的软件设计环境和开发工具，如进行模拟或仿真。随着 VLSI 迅速发展，硬件价格不断下降，软件却日益复杂，其设计时间和费用不断增加，加上对软件基本模块操作不断深入认识，使软、硬界面有上升的趋势，即有更多的软件功能由硬件完成，或者为软件提供更多的硬件支持。

系统结构设计步骤如下：

(1) 需求分析。对该系统的应用环境(实时处理、分时处理、网络、远程处理、事务处理、科学计算、容错、高保密性等),所用语言种类和特性,对操作系统要求,所用外部设备的特性,进行技术经济分析和市场分析。

(2) 需求说明。包括设计准则(造价、可靠性、可行性、可扩性、兼容性、速度、安全性、灵活性),功能说明,所用芯片说明,引入新结构成功的把握性以及程序设计方便性等。

(3) 概念设计。根据上述已确定的准则进行软、硬件功能分析,确定机器级界面。

(4) 具体设计。对机器级界面的各个方面进行详细、具体的定义,包括数据表示、指令系统、寻址方式、存储结构、中断系统、I/O 方式、总线结构等。该阶段可提出几种方案供选择。

(5) 设计优化和评价。可反复进行,以期获得尽可能高的性能价格比。

1.5 现代计算机系统结构的研究领域

1.5.1 计算机系统结构分类

计算机系统的基本工作过程是执行一组指令序列,对一组数据进行处理,因此 Michael. J. Flynn 于 1966 年提出了基于指令流和数据流的多倍性的计算机系统结构分类方法。

关于计算机系统结构有下列名词术语:

(1) 指令流 (Instruction Stream) —— 机器执行的指令序列。

(2) 数据流 (Data Stream) —— 由指令流调用的数据序列(包括输入数据和中间结果)。

(3) 多倍性 (Multiplicity) —— 在系统最受限的部件上,同时处于同一执行阶段的指令或数据的最大个数。

按指令流所具有的多倍性, Flynn 将计算机系统结构分类如下:

(1) 单指令流单数据流 (SISD)

(2) 单指令流多数据流 (SIMD)

(3) 多指令流单数据流 (MISD)

(4) 多指令流多数据流 (MIMD)

图 1-11 分别表示了它们的基本结构(不包括 I/O 设备)。

1. SISD系统结构

这种系统结构表示了大多数串行计算机,如图 1-11 (a) 所示。指令顺序执行,在指令执行阶段如采用流水线处理则可有重叠。在这类结构中,有的可能设置多个并行存储体和多个执行部件。但是只要指令部件一次只对一条指令进行译码,并且只对一个执行部件分配数据,则它仍属于 SISD 类。所以 SISD 单处理机系统可以是流水线的,可以有一个以上的功能部件,所有功能部件均由一个控制部件管理。

2. SIMD系统结构

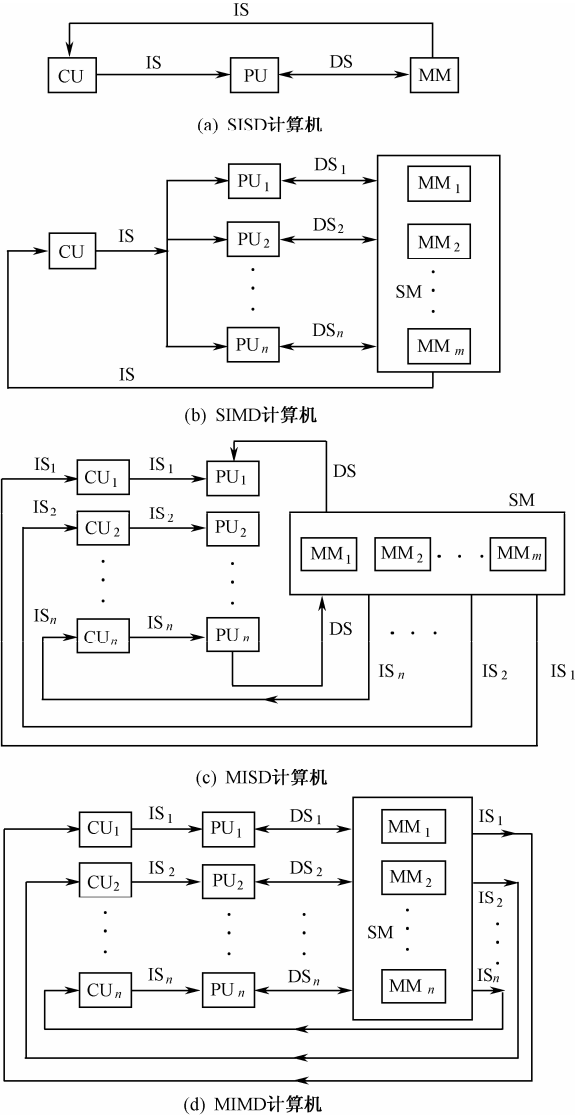
这种系统结构以并行处理机(阵列处理机)为代表,如图 1-11 (b) 所示。在同一个控制部件管理下,有多个处理单元 PU,所有 PU 均接收从控制部件送来的同一条指令,但操作对象却来自不同数据流的数据组,共享存储器的子系统可以有多个模块。这类计算机还包括相联处理机。从处理数据的并行性角度分析,还可分成位片式(位串行字并行)和字片式(位、字全并行)。

3. MISD系统结构

图 1-11（c）说明了这种计算机的结构。系统中共有 n 个处理单元 PU，各配有相应的控制部件 CU。每个处理单元接收不同的指令，但运算对象是同一个数据流及其派生数据流（如中间结果）。在这个宏观流水线中，每个处理器的输出结果是下一个处理器的输入操作数。这类系统结构无实用价值，目前没有实际的机器与之对应。

4. MIMD系统结构

大多数多处理机系统和多计算机系统可以划归这一类结构，如图 1-11（d）所示。一台内在的 MIMD 计算机意味着 n 个处理机之间存在相互作用，因为所有数据流均来源于由全体处理机共享的同一个数据空间。如果 n 个数据流来源于共享存储器的不相邻子空间，则可视作为多 SISD（MSISD）操作，这只不过是 n 个独立的 SISD 单处理机系统的集合，是松散耦合型。一个内在的 MIMD 系统，其处理机之间相互作用程度很高，是典型的紧密耦合型。



CU—控制部件；PU—处理部件；MM—存储器模块；SM—共享存储器；IS—指令流；DS—数据流

图 1-11 Flynn 分类法的系统结构

Flynn 分类法反映了大多数计算机的并行型、工作方式和结构特点。但是基本上是对冯·诺依曼型计算机进行分类,对非冯·诺依曼型计算机(如数据流计算机)没有包括进去。而对冯·诺依曼型的流水线处理机划归哪一类,现在仍有不同意见。有人认为,把标量流水线处理机划入 SISD,向量流水线处理机划入 SIMD 比较合适。

1972 年,美籍华人冯泽云(Tse-yun Feng)提出用最大并行度(Degree of Parallelism)对计算机系统结构进行分类。所谓最大并行度 P_m 是指计算机系统在单位时间内能够处理的最大的二进制位数。设一个时钟周期 Δt_i 内能处理的二进制位数为 P_i ,则 T 个时钟周期内平均

$$P_a = \frac{\sum_{i=1}^T P_i}{T}$$

并行度

平均并行度 P_a 取决于系统的运行情况,与应用程序有关。因此系统在 T 个时钟周期内的

$$\mu = \frac{P_a}{P_m} = \frac{\sum_{i=1}^T P_i}{TP_m}$$

平均利用率

最大并行度 P_m 定量地反映了对数据处理的并行性。

$$P_m = nm$$

式中, n 是同时处理时一个字中的二进制位数; m 是能同时处理的字数。

按计算机对数据的处理方式划分,则 P_m 值有下列 4 种类型:

- (1) 字串位串 (WSBS), 其 $n=1, m=1$;
- (2) 字串位并 (WSBP), 其 $n>1, m=1$;
- (3) 字并位串 (WPBS), 其 $n=1, m>1$ (即位片处理);
- (4) 字并位并 (WPBP), 其 $n>1, m>1$ (即全并行处理)。

Wolfgang Handler 根据计算机系统硬件结构的并行程度和流水线处理程度进行分类,着重于处理器控制部件(PCU)、算术逻辑部件(ALU)和位级电路(Bit-Level Circuit, BLC)的“并行-流水线”处理。PCU 可视作一个处理机或一个 CPU, ALU 相当于 SIMD 的处理单元(PE), BLC 对应于在 ALU 中进行一位运算所需的组合逻辑电路。

一个计算机系统 C 可由 6 个独立项组成的三元组表征,其定义如下:

$$T(C) = (K \times K', D \times D', W \times W')$$

式中, K 是 PCU 数; K' 是可组成流水线的 PCU 数; D 是 ALU (或 PE) 数; D' 是可组成流水线的 ALU 数; W 是 ALU (或 PE) 的字长; W' 是一个 ALU (或 PE) 中的流水线段数。

例如, Texas Instrument 公司的 ASC 系统,有一个控制器,控制 4 条运算流水线,每条流水线有 64 位长,共分 8 段,则

$$T(ASC) = (1 \times 1, 4 \times 1, 64 \times 8) = (1, 4, 64 \times 8)$$

又例如, CDC 公司的 6600 系统,有一个 CPU,一个 ALU 有 10 个功能部件,每个字长为 60 位,10 个功能部件组成一条流水线。同时, CDC-6600 有 10 个并行工作的 I/O 处理器,每个 I/O 处理器有一个 ALU,其字长为 12 位,则

$$T(CDC6600) = T(\text{中央处理机}) \times T(\text{I/O 处理机}) = (1, 1 \times 10, 60) \times (10, 1, 12)$$

Flynn 分类法的典型产品见表 1-1 所示。Feng 氏分类法的典型产品见表 1-2 所示。Handler 分类法的典型产品见表 1-3 所示。

表 1-1 Flynn 分类法产品

类 别		型 号
SISD	用单个功能部件	IBM701, IBM1401, IBM7090, PDP-11, VAX-11/780
	用多个功能部件	IBM360/91, IBM370/168UP, CDC6600, B-5000
SIMD	数据全并行处理	ILLIAC-IV, PEPE, BSP, STAR-100, TI ASC, AP-120B, IBM3838, CRAY-1, CYBER-205, VP-200, CDC-NASF
	数据位片串行处理	STARAN, MPP, DAP
MIMD	松耦合	IBM370/168MP, UNIVAC1100/80, IBM3081/3084, C_m^*
	紧耦合	Burroughs D-825, Cmpp, CRAY-2, S-1, CRAY-XMP, HEP

表 1-2 Feng 氏分类法产品

处理方式	机器型号与并行度
WSBS	$P_m(\text{EDVAC}) = (1, 1)$
WSBP	$P_m(\text{IBM370/168}) = (32, 1)$, $P_m(\text{CDC6600}) = (60, 1)$
	$P_m(\text{B7700}) = (48, 1)$, $P_m(\text{PDP-11}) = (16, 1)$
	$P_m(\text{VAX-11/780}) = (32, 1)$, $P_m(\text{Z-80}) = (8, 1)$
WPBS	$P_m(\text{STARAN}) = (1, 256)$, $P_m(\text{MPP}) = (1, 16384)$, $P_m(\text{DAP}) = (1, 4096)$
WPBP	$P_m(\text{ILLIAC-IV}) = (64, 64)$, $P_m(\text{TI ASC}) = (64, 32)$
	$P_m(\text{Cmpp}) = (16, 16)$, $P_m(\text{S-1}) = (36, 16)$

表 1-3 Handler 分类法产品

计算机型号 $T(C)$	系统说明 ($K \times K'$, $D \times D'$, $W \times W'$)
$T(\text{TI-ASC})$	(1, 4, 64×8)
$T(\text{CDC-6600})$	(1, 1×10, 60) × (10, 1, 12) 中央处理器 I/O 处理器
$T(\text{ILLIAC IV})$	(1, 64, 64)
$T(\text{MPP})$	(1, 16384, 1)
$T(\text{C.mmp})$	(16, 1, 16) + (1×16, 1, 16) + (1, 16, 16)
$T(\text{PEPE})$	(1×3, 288, 32)
$T(\text{IBM 360/91})$	(1, 3, 64×(3~5))

1.5.2 现代计算机系统结构研究方向

由于科学技术的进步，计算机技术的应用领域不断扩大与深入，对计算机体系结构的研究与开发提出了新的要求，原有的冯·诺依曼型结构，已经不能满足要求。计算机系统结构的研究与设计
 和计算机技术应用的扩大与深入、VLSI 技术新进展、算法与软件研究、硬件价格下降等，使计算机系统的设计者不断采用新的器件，改进算法，增加硬件辅助部件，设计并行系统结构等技术，来提高系统的性能，使系统的性能得到较大改善。计算机系统结构的研究工作，正向着多种高性能方向发展，当前的研究主要集中在如下 9 个方面。

- (1) RISC 结构;
- (2) Client/Server 结构;
- (3) Mpp 矢量模式(Vector Mode);
- (4) Massively Parallel Processing 并行处理;
- (5) Cluster Computing 网络型并行计算 (Networked Parallel Computing, NPC);
- (6) Network 结构;
- (7) 数据流计算机结构;
- (8) 分布计算环境结构;
- (9) 网格。

现代计算机系统结构组成的基本模式如图 1-12 所示。

本书仅就计算机系统结构的几种主要类型进行较深入探讨，对计算机原理及其他有关内容本书不再重复。

1.5.3 计算机系统结构发展趋势

计算机系统结构向并行处理方向发展是一个必然趋势。影响并行计算机体系结构的有三个因素：过去和当前发展趋势所揭示的传统力量；阻碍将来的进程沿原趋势发展的基本限制；中止当前发展并形成新趋势的突破。按照并行性，1998 年形成的并行计算机市场金字塔如图 1-13 所示。单处理器 PC、工作站和服务器的台数为千万到亿的数量级。2~4 个并行处理器的计算机台数为 10 万到数百万的规模，这些一般都是专用服务器。而 5~30 个的处理器的主要是专用的高端服务器。在数十到上百的处理器规模上的机器主要是支持大规模数据库、大型科学计算或者大工程设计（如石油勘探、结构模型、流体动力等）的专用计算机。20 世纪 90 年代以来，出现了 1000 到几万个处理器的高性能计算机系统。

目前处理器的性能每十年增加 100 倍，而组成存储器的主要芯片 DRAM 容量也是每十年增加 100 倍，因此并行机器中节点计算性能与存储容量（MFLOPS/MB）可保持基本平衡。但是微处理器的时钟频率以每十年 10~15 倍的速度增长，集成的晶体管以每十年 30 倍的速度增长。另一方面，DRAM 存取周期的时间改善却非常慢，每十年大约以两倍的速度改进。因此，处理器速度与存储器速度间的差距在继续加大。为了保持处理器的速度增长趋势，在两者速度差距加大的前提下，要求处理器采用更好的避免时延技术和容许时延技术。由于周期时间和并行性的组合，处理器指令速率的增加将对存储器的带宽提出更高要求。由于避免时延和高存储带宽以及芯片上存储容量的增加将导致存储层次结构更加复杂，同时使指令级并行性动态调度增加。存储结构每增加一层都会增加访存的开销，提高命中率 and 缩短命中时间比减少失效时的损失更有效。失效损失也是通信开销的一部分，先进的体系结构和良好的编程能够将不必要的通信量减少到最小。多线程在处理器中得到广泛应用，通过在单处理器内引入线程级并行，从而减少从一个处理器到多个处理器的传输开销，使小规模 SMP（机群系统处理）更具吸引力。

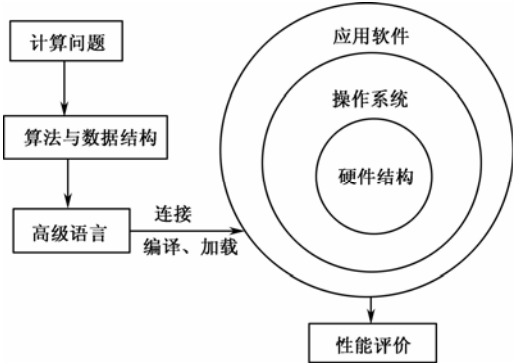


图 1-12 现代计算机系统结构组成的基本模式

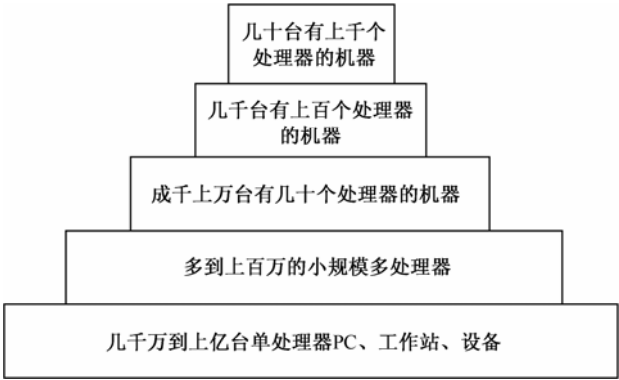


图 1-13 并行计算机的市场金字塔

构建并行计算机体系结构的重要部分之一是通信网络，选择光纤是构建高速链路的必由之路。千兆位以太网使用光纤为物理链路，具有小物理截面的灵活的高性能光纤已成为相当好的链路技术，通过提高光纤收发器的数量可使成本下降。许多系统设计已通过要求所有部件传输整个高速缓存(Cache)的行来使总线效率更高。I/O 设备接口(适配器)也要按照 Cache 的一个块来构造，同时还要支持 Cache 一致性协议。这样，从本质上讲，所有系统都是 SMP，哪怕只连接一个处理器。总线逐渐地被交换机所替代，总线侦听协议(CSMA/CD 协议)逐渐被目录替代。即使使用总线，也采用分组交换技术。高性能的互连网络已在大量并行计算机设计中被采用。更高速的网络也是 I/O 子系统的主要问题，如 PCI 总线已经取代了传统的 ISA, EISA, MCA, VME, MUITBUS 等总线，标准 PCI 总线可提供 1Gb/s 的带宽，对于扩展到 64 位的 PCI 总线，具有 66MHz 的操作速度。从这些趋势可以断定，机群结构以及紧密组装的商用节点集将成为构建大规模系统的必然选择。随着高性能互联网更加成熟和继续改进,Cache 接口协议和 Cache 一致性协议会更好集成，由此大规模并行计算机更多使用 SMP 节点，SMP 群将会成为并行计算的工具。

并行计算机体系结构发展可能遇到三种障碍：时延、开销、成本或功耗。时延障碍本质上是光的传输速度或电子信号的传输速度问题。时延确实存在，但可预见不会成为并行计算机体系结构发展的严重障碍。这是因为，容许时延技术在处理器级数十个周期上是非常有效的。其次，存储操作采用流水线形式，而且允许处理器从指令窗口发射多条指令，此时，转移指令的精确判断将比时延更为重要，若预测成功，可提供 100 个周期左右的存储器访问时间。同时，多线程技术为指令级并行提供了隐藏时延以及非成功预测转移开销的可能性。目前很多通信时延是在网络接口上(尤其是存储转发时延发生在源和目的节点)，而不是网络本身。网络接口时延已经占据网络时延的较大部分，该时延随着网络接口发展而减少。

另外，存储结构中每层都要增加数据传送的开销，因为数据必须经过所有层的接口。片内 Cache 与片外 Cache 间的接口开销会大一些，然而经过存储总线传送数据时，由于复杂的协议带来更大的开销。降低开销的方法是增加一次传输的最小数据量，增加最小信息块的 Cache 行的大小，达到字节开销(即总开销/一次传送字节数)下降的目的。

Little 定律揭示了成本障碍。该定律认为，若需要隐藏的总时延为 L ，长时延请求的发生率为 ρ ，则当该时延被隐藏时每个处理器需要的在线请求数为 ρL ， P 个处理器需要的网络带宽为 $P\rho L(P)$ 。 $L(P)$ 反映了随机器增大而时延增加。为了控制时延增加，网络总带宽就需要提高，从

而网络成本增加，产生了成本障碍。因此，在网络技术上需要考虑带宽和成本间的协调。

在 20 世纪 80 年代中期，微处理器（MPU）突破了关键技术，出现了 32 位 MPU，具有一定性能和容量的完整的计算机放在了一个电路板上，多个这样的电路板组成一个系统。基于总线 Cache 一致性的 SMP 替代了大规模数据中心、事务处理、工程分析。基于 MPU 的大规模并行系统替代了向量超级计算机。同时，微处理器和存储器（主要是 Cache）集成在一个芯片上，并不断提高其集成度，表 1-4 反映了 MPU 中 Cache 占用的晶体管数和面积的百分比。

表 1-4 MPU 中 Cache 占用晶体管数和面积的百分比

年份	微处理器	片载缓存大小	晶体管总数(个)	用于存储器的百分比	芯片面积 (mm ²)	用于存储器 的百分比
1993	Intel Pentium	I: 8 KB D: 8 KB	3.1 M	32%	~300	32%
1995	Intel Pentium Pro	I: 8 KB D: 8 KB L ₂ : 512 KB	P: 5.5 M +L ₂ : 31 M	P: 28% +L ₂ : 100% (Total: 88%)	P: 242 +L ₂ : 282	P: 23% +L ₂ : 100% (Total: 64%)
1994	Digital Alpha 21164	I: 8 KB D: 8 KB L ₂ : 96 KB	9.3 M	77%	298	37%
1996	Digital Strong-Arm SA 110	I: 16 KB D: 16 KB	2.1 M	95%	50	61%

注：I—指令，D—数据，P—处理器晶片，L₂—二级高速缓存

处理器和存储器集成出现了新的概念——PAM（Processor And Memory）。表 1-4 中的片载缓存是指用处理器相同工艺制作的 SRAM（静态 RAM）作为 Cache。而 DRAM（动态 RAM）使用的是完全不同的工艺，因此将处理器与 DRAM 集成是 PAM 的新发展，中等规模 PAM 芯片将提供更大的处理能力和存储容量，更便于组成系统，图 1-14 表示计算机系统的各种带宽。处理器数据通路为几个字宽，通常它与 L₁ 缓存的接口为两个字。L₁ 缓存的块的容量是 32 或 64 字节，其制约条件是处理器每次一个字的处理方式和 MPU 芯片界面。L₂ 缓存的容量更大些，制约条件是处理器芯片的接口和存储器总线的接口。形成一个存储模块的 SIMM 接口比存储总线要宽但要慢。在 DRAM 内部，有一个很宽的数据缓冲区，直接与位阵列交换数据。

关于 DRAM 的结构，有两种方案。一种是在常规 DRAM 芯片数据缓冲区的逻辑上相关部位加入简单的专用处理单元，如图 1-15 所示。这种方法称为存储器中的处理器 PIM（Processor In Memory）和计算 RAM。其实质上是一个数据并行操作的受限类型的 SIMD 处理。另一种是加强对 DRAM 体中数据缓冲区的支持，作为向量寄存器用，如图 1-16 所示。当使用宽的位阵列时，在 DRAM 行与向量寄存器之间就可实现宽带传输，但是对向量寄存器的算术运算是通过数据流经少数传统功能部件实现的。革命性的方案是去掉 DRAM 与处理器之间的片外 Cache 和存储系统之间所有层次，与 DRAM 有关的数据缓冲作为最后一层 Cache，实现通用设计，DRAM 集成于处理器芯片内，DRAM 数据缓冲区的 Cache 可与一个或多个在同一芯片的处理器进行数据传输。

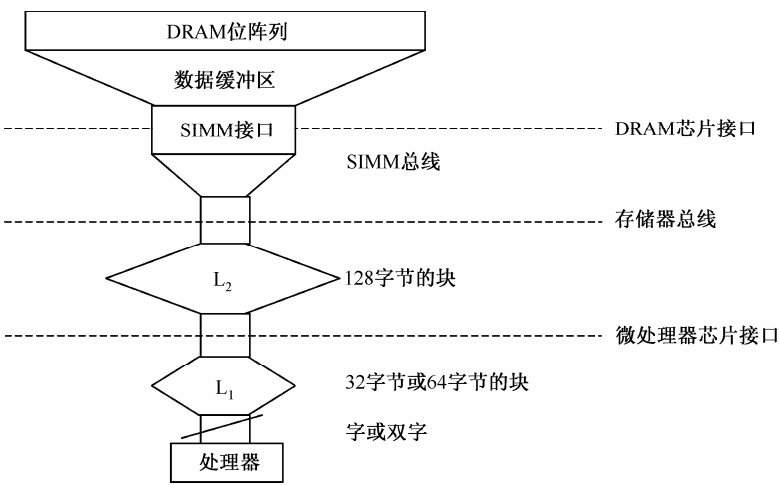


图 1-14 计算机系统的各种带宽

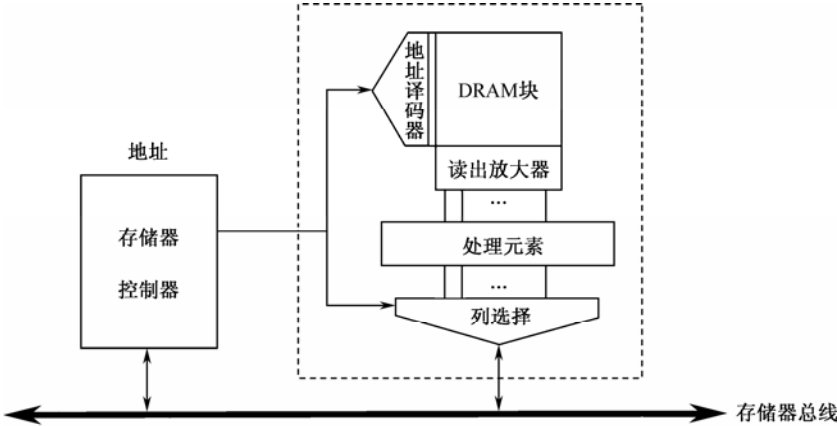


图 1-15 存储器中的处理器方案

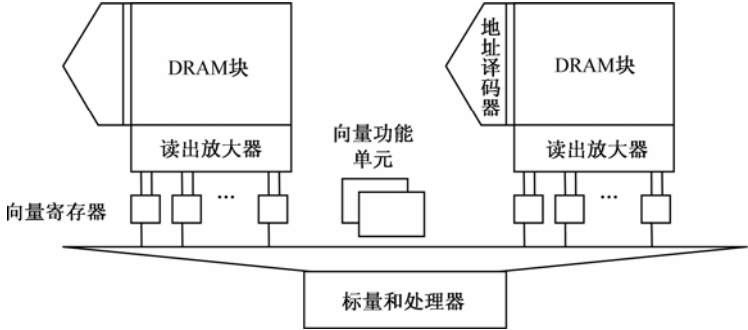


图 1-16 向量 DRAM 组成方案

第 2 章 计算机系统结构的合成

计算机系统结构设计是以机器语言级界面开始的，进行软、硬件功能分配。本章以该界面为起始点，对机器语言级机器的硬件结构进行提纲挈领的阐述，对中央处理器、总线结构、存储系统和 I/O 系统的基本结构和基本概念给出简明扼要的说明，为读者以后章节学习奠定基础。

2.1 中央处理器

计算机系统的核心硬件是中央处理器 CPU（Central Processing Unit），它由控制器、运算器和寄存器组成，通常称为处理器（Processor）。利用超大规模集成电路制成的处理器芯片称为微处理器（Microprocessor）。处理器的任务是取出指令、解释指令和执行指令。为此，每种处理器都有自己的一套指令。

2.1.1 CPU组成

如图 2-1 所示，如果处理器和内存存储器 ROM，RAM 结合在一起，称为中央电子集合体（Central Electronic Complex，CEC）。

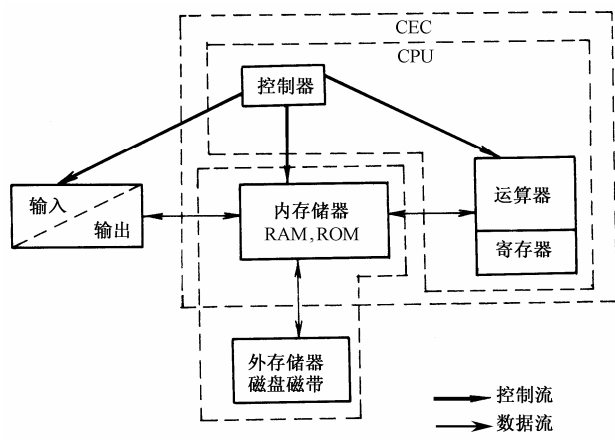


图 2-1 CPU 组成

CPU 中三个部件的功能如下：

- (1) 控制器。其任务是负责从存储器中取出指令，确定指令的类型，并对指令进行译码，控制整个计算机系统一步一步地完成各种操作。
- (2) 运算器。为计算机提供计算与逻辑功能，控制器把数据带给 ALU 后，就根据指令完成算术运算或逻辑判断及逻辑运算。所谓算术运算，就是加、减、乘、除；所谓逻辑判断，通常是根据比较的结果进行选择，如比较两个数是否相等，如果相等则继续处理，如果不相等则继续处理，如果相等

则停止处理。所谓逻辑运算是指与、或、非、异或等。

（3）寄存器（Register）。它是处理器内部的存储单元，控制器中的寄存器，用于保存程序运行中的状态，存储当前指令信息和将要执行的下一条指令的地址等。运算器中的寄存器，用于存储进行运算与比较的数据及其结果。当然，寄存器的容量非常有限，远远放不下处理器运行某一程序所需的全部信息。这些信息就存储在 RAM，ROM 等内存中，如果连内存也放不下，就只好留在外存中。事实上，程序原来都是放在外存中的，根据运行需要才调入内存存储器，直到执行某条指令之前，它才送入寄存器。

2.1.2 数据表示

1. 数据结构与数据表示

计算机常用的各种数据结构有串、堆栈、队列、向量、阵列、矩阵、链表、图等。实现这些数据结构的各种算法，几乎都立足于计算机系统结构只提供按地址访问的一维线性存储器以及最基本、最简单的数据表示。数据表示指能由硬件直接辨认的数据类型。这样，这些数据结构要经过软件映像，变换成按地址访问一维存储器内的各种数据表示。如何用最少的存储空间存储这些数据结构，以及用什么样的算法能最快、最简地存取，则是数据结构的研究课题。

计算机的基本数据类型有逻辑（布尔）数、定点数（整数）、浮点数（实数）、十进制数、字符串和数组等。对这些数据的运算可以设置专门指令；或仅设置最简单的算术逻辑运算指令，而通过编程并执行实现它，但速度下降很多。在机器中若设置能直接对矩阵向量数据（数组）进行运算的指令及硬件，可以大大提高对向量、数组的处理速度，这一般在巨型机中才使用。

目前计算机所用数据字长有 8，16，32 位等。在存储器中每个单元的容量为 8 位，即一字节，故存储器单元地址按字节编址。计算机的指令系统可支持对字节（8 位）、半字（16 位）、字（32 位）、双字（64 位）的运算。

计算机的数据表示如何确定，是一个复杂的问题。除了必不可少的基本数据表示之外，数据表示的确是一个软、硬件分配的问题。这是因为，各种数据结构本来是能够在只有最简单、最基本数据表示的机器中实现的，只是在有了更好的数据表示之后，实现的效率更高罢了。

因此，衡量某种数据表示是否合适的标准，首先要看它使执行时间和所需存储容量减少了多少。而衡量执行时间是否减少的一个重要的指标是看主存与处理机间所需传送的信息量是否减少。例如，有两个 200×200 元素（定点数）的阵列 A 和 B 进行相加运算，若用 PL/I 语言仅需一条语句，经优化编译形成 6 条机器指令的目标程序，其中有 4 条需循环执行 40 000 次。但若机器有阵列型数据表示，则只需一条机器指令就能执行两个阵列相加的运算，所以只需一次访问存储器操作，处理机与主存间取指操作减少了 $4 \times 40\,000$ 次，故执行时间大大缩短。

衡量某种数据表示是否合适的另一个标准，是看这个数据表示的通用性如何，利用率如何。对于基本数据表示，其通用性和利用率当然不构成任何问题，而对于上述阵列机的数据表示（指阵列数据表示），则其通用性、利用率就不理想了。而且，阵列数据表示还存在能表达多大阵列的问题，阵列过大，硬件投资增加，机器性能价格比下降，利用率可能不高；阵列过小，阵列运算要拆成多块，分批进入阵列运算部件，使编译产生困难。所以机器的数据表示的选择应该综合平衡上述各种因素后才确定。

2. 定点数选择

常见的定点数有两种：一是纯整数，如图 2-2 (a) 所示；二是纯小数，如图 2-2 (b) 所示。

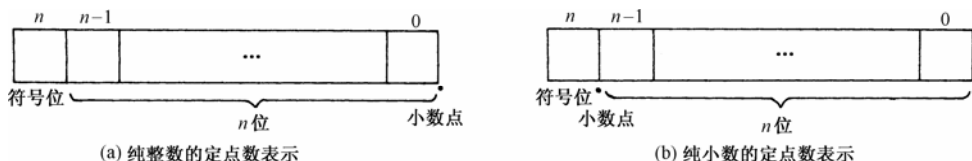


图 2-2 定点数的表示

定点数的正、负用符号位表示，符号位为 1 表示负数，常用补码；符号位为 0 表示正数，是原码。下面讨论定点数表示的数值范围。

(1) 纯整数的定点数，基值 $r_m=2$ 时（二进制）。

最小值为 $N_{\min} = -2^n$ ($n \geq 1$ 为正整数)

最大值为 $N_{\max} = 2^n - 1$

可表示的个数为 $2 \times 2^n = 2^{n+1}$

(2) 纯小数的定点数，基值 $r_m=2$ 时（二进制）。

最小值为 $N_{\min} = -2^{-n}$ ($n \geq 1$ 为正整数)

最大值为 $N_{\max} = 1 - 2^{-n}$

可表示的个数也为 2^{n+1} 个 (n 是有效数值位个数)。

例如，纯整数定点数有效数值位 $n=7$ ，带符号位为 8 位（1 字节），则所能表示的最小数为

$$N_{\min} = -2^7 = -128$$

最大数为 $N_{\max} = 2^7 - 1 = 127$

可表示的数的个数为 $2^{n+1} = 2^8 = 256$

3. 浮点数选择

任意进位制的浮点数所表示的数字范围表述如下：当基值 $r_m=2$ 时，阶码为 p 位，尾数为 m 位，其最小绝对值和最大绝对值为

$$|N|_{\min} = 2^{-(2^p-1)} \times 2^{-m}$$

$$|N|_{\max} = 2^{(2^p-1)} \times (1 - 2^{-m})$$

可表示值的范围为 $2^{-(2^p-1)} \times 2^{-m} \leq |N| \leq 2^{(2^p-1)} \times (1 - 2^{-m})$

对 r_m 为任意制（即任意进位制），尾数为 m' 位，阶码的基 $r_p=2$ （仍为二进制），阶码为 p 位，则可表示值的范围为

$$r_m^{-(2^p-1)} \times r_m^{-m'} \leq |N| \leq r_m^{(2^p-1)} \times (1 - r_m^{-m'})$$

若尾数、阶码均为任意基值（即为任意进位制），则上式为

$$r_m^{-(r_p^{p'}-1)} \times r_m^{-m'} \leq |N| \leq r_m^{(r_p^{p'}-1)} \times (1 - r_m^{-m'})$$

式中， r_m 为尾数基， m' 为尾数位数， r_p 为阶码基， p' 为阶码数位（上述公式是以“符号-绝对值码制”为基础的）。

规格化浮点数表示如图 2-3 所示。该图表示的是二进制规格化浮点数格式（以“符号-绝对值码（即原码）”为基础）。

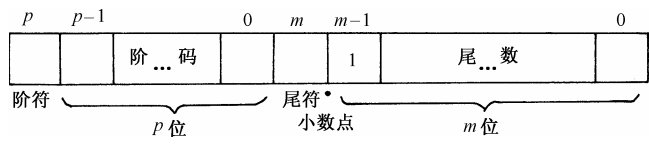


图 2-3 规格化浮点数的表示

在阶码相同的前提下，尾数采用不同基值时，表示的浮点数大小及个数均不相同。现以阶码 $p=2$ 且正阶、正尾浮点数为例，比较 $r_m=2$ 和 $r_m=16$ 时的不同情况，如图 2-4 所示，在尾数基值 $r_m=2$ （二进制）， $r_p=2$ ， $m'=4$ （即尾数 bit3~bit0，共 4 位）、 $p'=2$ 时作规格化表示（即此时 bit3=1）的正阶、正尾的数值范围如下：

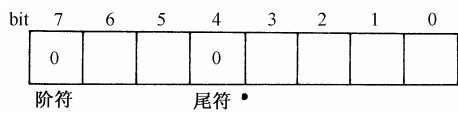


图 2-4 正阶、正尾浮点数的表示

最小尾数为

$$r_m^{-1}=2^{-1}=\frac{1}{2}=\frac{8}{16}$$

最大尾数为

$$1-r_m^{-m'}=1-2^{-4}=\frac{15}{16}$$

最小阶码为

$$2^0-1=0$$

最大阶码为

$$2^{p'}-1=2^2-1=3$$

阶码的个数为

$$2^{p'}=2^2=4$$

可表示的最小正浮点数为

$$r_m^{-1}\times r_m^0=\frac{8}{16}=\frac{1}{2}$$

可表示的最大正浮点数为

$$(1-r_m^{-m'})\times r_m^{(2^{p'}-1)}r_m^{(2^{p'}-1)}=7\frac{1}{2}=\frac{15}{2}$$

可表示的正阶正尾规格化的数的个数为

$$2^{p'}\times 2^{m'}\times \frac{1}{2}=32$$

式中， $2^{m'}$ 表示的尾数位数应能表示二进制数的个数；1/2 是因为规格化，取其个数的一半（bit3 必为 1）；乘 $2^{p'}$ 是因阶码扩大而增加的倍数。

在尾数基值 $r_m=16$ （十六进制）， $r_p=2$ ， $m'=1$ （尾数 bit3~bit0 四位二进制组成一个十六进制数）、 $p'=2$ 时，作规格化表示（即 bit3~bit0 中必须有一位为“1”，不许出现全“0”，bit3~bit0 为 0001~1111）的正阶、正尾数的范围如下：

最小尾数为

$$r_m^{-1}=16^{-1}=\frac{1}{16}$$

最大尾数为

$$1-r_m^{-m'}=1-16^{-1}=\frac{15}{16}$$

最小阶码为

$$2^0-1=0$$

最大阶码为

$$2^{p'}-1=2^2-1=3$$

阶码的个数为

$$2^{p'}=2^2=4$$

可表示的最小正浮点数为 $r_m^{-1} \times r_m^0 = \frac{1}{16}$

可表示的最大正浮点数为 $(1 - r_m^{-m'}) \times r_m^{(2^{p'} - 1)} = (1 - 16^{-1}) \times 16^3 = 3840$

可表示的正阶正尾规格化的数的个数为

$$2^{p'} \times r_m^{m'} \times \frac{15}{16} = 2^2 \times 16^1 \times \frac{15}{16} = 60$$

式中, $r_m^{m'}$ 表示基值为 r_m 时, 共 m' 位应表示的个数; $15/16$ 是因为十六进制规格化时, 只有 0000 不是规格化数, 故取其 $15/16$; 乘 $2^{p'}$ 是因阶码扩大而增加的倍数。

从上述例子, 可得到下列两个结论:

(1) 对于相同阶码与尾数位数, r_m 大, 则表示数的范围也大。 r_m 只能使用 2, 4, 8, 16…。一般使用 2, 8, 16 三种。

(2) r_m 大时, 虽然表示数的范围大, 表示数的个数增加, 但在数轴上分布较稀疏。如上例, 在数轴 $(1/2) - 2$ 之间, $r_m=2$ 时, 有数 15 个; $r_m=16$ 时, 有数 8 个。

形成上述情况的原因有两点: 一是因为采用规格化表示的缘故, 如上例中 $r_m=2$ 时, 规格化后, 其尾数个数只为原来的 $1/2$; 二是因为在同长度阶码时, r_m 不同, 每次小数点移动位数不同, 上例中 $r_m=2$ 时, 1 个阶码值移动小数点 1 位, 而 $r_m=16$ 时, 则移动 4 位。故可以归纳如下: r_m 增大, 表示数的个数增加, 数轴上分布稀疏, 从而计算误差增加。

对于正阶、正尾规格化数的全部表示, 归纳为如下公式:

最小尾数为	r_m^{-1}
最大尾数为	$1 - r_m^{-m'}$
最小阶码为	0
最大阶码为	$2^{p'} - 1$
阶码个数为	$2^{p'}$
最小浮点数为	$r_m^0 \times r_m^{-1}$
最大浮点数为	$r_m^{(2^{p'} - 1)} \times (1 - r_m^{-m'})$
表示数的个数为	$2^{p'} \times r_m^{m'} \times \frac{r_m - 1}{r_m}$

4. 浮点数的下溢处理

浮点数的右移操作会把有效位移掉而造成精度损失。精度损失主要来源于中间运算过程会使信息长度增长 (如在两数相乘时), 以至超过运算器和存储器的字长表示范围, 不得不截断。当然, 可以用两倍字长运算来解决, 但是又增加了存储空间和运算时间。因此, 对于一般的应用, 要采取措施尽可能地减少运算过程中带来的精度损失, 关键是处理好尾数下溢。

尾数下溢主要产生于加法中的对阶、规格化右移以及乘法中取单倍长度的乘积。常用的处理办法有如下 4 种:

(1) 截断法。也称不舍入法, 就是简单地将下溢部分截去。截断所带来的误差, 在整数运算中最大可接近于 1 (如 7.999 截断到 7), 在分数运算中接近于 $r_m^{m'}$ (如二进制分数运算则接近于 $2^{-m'}$)。这些误差大都是负误差, 少数是无误差截断, 且其概率分布是均匀的。

(2) 舍入法。被截尾数为 1 进 1, 其误差有正、有负, 但总的趋向是稍偏于正误差, 平均误差比截断法小, 最大误差小于一半。

(3) 恒置“1”法。对被截断的尾数最低位，无论原先是“0”还是“1”，都恒置成“1”。舍入法在最坏情况下可能需从尾数最低位向最高位的进位时间，甚至会发生上溢而要进行右移规格化。而恒置“1”法和截断法一样不需要额外处理时间，然而其最大误差比截断法大。因为，当尾数舍去部分为全“0”时，如 10.000…0 时，截断法尾数为 10，而恒置“1”法尾数为 11，产生尾数最低位为“0”误为“1”的误差。

(4) ROM 或 PLA 舍入法。也称查表舍入法，用只读存储器或组合逻辑实现，其工作原理如图 2-5 所示。

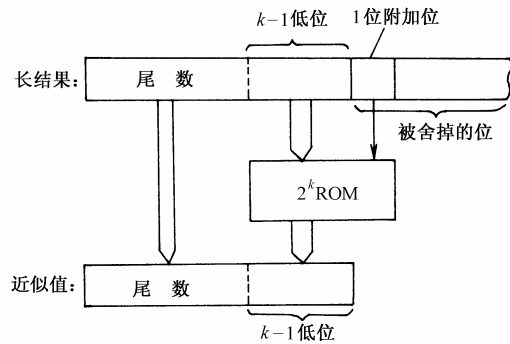


图 2-5 ROM 舍入法原理

图 2-5 中 2^k 容量的 ROM 中存放下溢处理用的表，其 k 位地址由尾数的最低 $k-1$ 位和一位附加位组成。一般情况下，按舍入法编码的 ROM 输出即为尾数最低 $k-1$ 位的下溢处理结果；当尾数的最低 $k-1$ 位为全“1”时，则采用截断法形成下溢处理结果，即仍输出 $k-1$ 位为全“1”。这样截断所产生的负误差正好补偿了其他情况下采用舍入法所产生的正误差，使平均误差接近于零。它集中了上述各种下溢处理方法的优点，而且避免了舍入法所需的相加进位的传输时间。

由上述分析可以看出，截断法最简单，速度最快，但平均误差最大。ROM 法比舍入法快，能调节平均误差使之接近于零，但硬件增加了。

5. 浮点数尾数基值的选择

浮点数表示数的范围、数的精度和数的效率都与尾数的基值 r_m 有关。而计算机中浮点数的总字长通常是确定的，除了尾数符号和阶码符号各占一个二进制位之外，其余就是尾数位数 m' 和阶码位数 p' 之和。所以，设计一种浮点数表示方式的关键是要解决两个问题：一是在浮点数总字长给定的情况下，如何选择尾数基值 r_m ，使浮点数表示的数的范围、精度、效率达到最高；二是在 r_m 确定后，如何根据表示的数的范围和精度确定尾数位数 m' 和阶码位数 p' 。

设有两种浮点数表示方式 F_1 和 F_2 ，它们字长相同，尾数都用原码或补码、纯小数表示，阶码都用补码或移码、整数表示，阶码的基数均为 2，而尾数的基数 r_m 不同。

F_1 : $r_{m1}=2$ ，尾数长度 m_1' ，阶码长度 p_1' ，则二进制字长 $L_1=m_1'+p_1'+2$ 。
 F_2 : $r_{m2}=2^k$ (k 为正整数)，尾数长度 m_2' ，阶码长度 p_2' ，二进制字长 $L_2=km_2'+p_2'+2$ 。
因为 F_1 与 F_2 的二进制字长相同，即 $L_1=L_2$ ，得

$$m_1'+p_1'=km_2'+p_2'$$

F_1 表示的数的范围为 $|N_1|_{\max} = 2^{2^{p_1'}}$

F_2 表示的数的范围为 $|N_2|_{\max} = (2^k)^{2^{p_2'}} = 2^{k \times 2^{p_2'}}$

又因为 F_1 与 F_2 表示的数的范围相同, 所以 F_1 与 F_2 的阶码应当相等, 即 $2^{p_1'} = k 2^{p_2'}$ 。两边分别取以 2 为底的对数, 得到 $p_1' = p_2' + \log_2 k$ 。把该式代入上式, 得到

$$\begin{aligned} m_1' + p_2' + \log_2 k &= k m_2' + p_2' \\ m_1' &= k m_2' - \log_2 k \end{aligned}$$

规格化浮点数精度主要与尾数基值 r_m 和尾数长度 m' 有关, 在一般情况下, 认为规格化尾数最后一位的精确度是一半, 这样规格化浮点数精度可以表示如下:

$$\delta(r_m, m') = \frac{1}{2} r_m^{-(m'-1)}$$

当 $r_m=2$ 时, $\delta(2) = \frac{1}{2} \times 2^{-(m'-1)} = 2^{-m'}$

F_1 的 $r_{m1}=2$, 所以 $\delta_1 = \frac{1}{2} \times 2^{(1-m_1')} = \frac{1}{2} \times 2^{(1-km_2' + \log_2 k)}$

F_2 的 $r_{m2}=2^k$, 所以 $\delta_2 = \frac{1}{2} \times 2^{k(1-m_2')}$

取 F_1 与 F_2 精度比值 $T = \frac{\delta_2}{\delta_1} = 2^{k - \log_2 k - 1}$

从上式可看出, 只有 $k=1$ (即 $r_m=2$) 或 $k=2$ (即 $r_m=4$) 时, $T=1$; 当 k 取其他任何值时, $T>1$, 表示 F_2 精度低于 F_1 。

由此可以得到如下结论: 在浮点数的字长和表示数的范围一定时, 尾数基值 r_m 取 2 或 4 时, 具有最高精度。

在浮点数的字长和表示数的精度一定时, 尾数基值 r_m 与表示数的范围之间有何关系呢? 由于 F_1 和 F_2 的精度相同:

$$\frac{1}{2} \times 2^{(1-m_1')} = \frac{1}{2} \times 2^{k(1-m_2')}$$

即 $m_1' = k m_2' - k + 1$

又因为 F_1 与 F_2 的二进制字长相同, $L_1=L_2$, $m_1'+p_1'=k m_2'+p_2'$, 将前式代入上式, 得

$$p_1' = p_2' + k - 1$$

因为 F_1 表示的数的范围为 $2^{2^{p_1'}} = 2^{2^{p_2'+k-1}} = 2^{2^{p_2'} \times 2^{k-1}}$

F_2 表示的数的范围为 $2^{k \times 2^{p_2'}}$

若假设 F_2 表示的数的范围大于 F_1 表示的数的范围, 则 F_2 阶码的最大值要大于 F_1 阶码的最大值, 即

$$2^{p_2'} \times k > 2^{p_2'} \times 2^{k-1}$$

即 $k > 2^{k-1}$

这个不等式在正整数定义域内无解, 即不存在比 F_1 表示的数的范围更大的浮点数表示方式, 只有当 $k=1$ ($r_m=2$) 或 $k=2$ ($r_m=4$) 时, F_2 阶码的最大值等于 F_1 阶码的最大值。由此可以得到另一个结论: 在浮点数字长和精度一定时, 尾数基值 r_m 取 2 或取 4 具有表示数的最大范围。

综合上述两个结论, 可以得出一个新的重要推论: 在浮点数字长确定后, 尾数基值 r_m 取

2 或 4 具有最大表示数的范围和最高精度。

浮点数表示方式的效率定义为

$$\eta = \frac{\text{可表示的规格化浮点数的个数}}{\text{全部浮点数个数}}$$

在忽略机器零后，有

$$\eta(r_m) = \frac{r_m - 1}{r_m}$$

当尾数基值 $r_m=2$ 时，

$$\eta(2) = \frac{1}{2} = 50\%$$

这是因为规格化浮点数表示中，尾数最高位有效值必须为 1，而尾数最高位有效值为 0 的浮点数是非规格化浮点数，不能列入，因此效率仅为 50%。

当尾数基值 $r_m=16$ 时，

$$\eta(16) = \frac{15}{16} = 94\%$$

与 $r_m=2$ 时相比，其效率提高了 1.875 倍。

从表示数的范围和精度角度出发， $r_m=2$ 或 $r_m=4$ 最好，但效率太差（ $r_m=2$ ，效率 $\eta=50\%$ ； $r_m=4$ ，效率 $\eta=75\%$ ），为了提高效率，在 $r_m=2$ 时，采用隐藏位表示方法，即规格化浮点数最高有效位一定是 1，在存储和传送时，最高位不表示出来，在计算时恢复出来或采用某种方法对运算结果予以修正，这时的效率可达 100%。 $r_m=4$ 不能采用隐藏位法，效率为 75%。

以前，IBM 公司的 360，370，4300 系列机 $r_m=16$ ；Burroughs 公司的 B6700，B7700 系列机 $r_m=8$ 。后来 DEC 公司的 VAC-11，Alpha，以及 CDC 公司的 6600，CYBER70 等和 Intel 公司的 x86 系列全部采用 $r_m=2$ 。浮点数尾数基值 $r_m=2$ 可达到最佳效果。

2.1.3 寻址方式分析

由于程序和数据存储于存储器中，所以寻址指如何确定数据地址及转移指令的下一条要执行的指令地址。每条指令的寻址方式由指令本身确定，或按某些预先约定的规则进行。有按地址访问、按堆栈访问、按内容访问等方式，还可以在指令中直接表示数据，称为立即数。与指令系统一样，不同的计算机的寻址方式也各不相同。

堆栈结构是现代计算机中不可缺少的组成部分，它有硬件和软件两部分。

硬件部分指：① 栈区（或称栈体），可由 CPU 中的寄存器组成，或为 RAM 中一个任选区域；② 堆栈指示器（SP），指向栈顶的空单元（或满单元），每次堆栈操作后，自动 ± 1 ；③ 后进先出（LIFO）控制电路，保证后进先出的存储操作。

软件部分指：① 单纯的堆栈操作指令，入栈指令（PUSH），出栈指令（POP）；② 子程序调用指令（CALL）和子程序返回指令（RET）；③ 外电路发中断请求，CPU 响应后，保护断点入栈。

按内容访问只适用于查询，在一般计算机中设置这样的指令尚不多见。按内容访问的相联存储器在“存储系统”一章中讨论。

按地址访问使用最普遍。对一般指令而言，是由机器指令指明（或经计算后得到）操作数据所在地址；对转移类指令而言，是下一条将执行的指令地址（或称程序转移的入口）。按地址访问有各种编址方式，在讨论它们之前，先介绍逻辑地址和物理地址的概念。

逻辑地址指程序员编程时所指定的程序（指令）地址和数据地址。物理地址指机器实际执行时的程序（指令）地址和数据地址。在早期，程序员编程时用的程序地址和数据地址（实

实际地址), 并不区分逻辑地址和物理地址, 但是随着多道程序以及汇编语言的使用, 很容易产生存储冲突, 再加上硬件结构上的一些特点, 如存储器容量可扩充性和采用虚拟存储器等, 决定了用户的源程序经编译后得到的是逻辑地址, 然后在程序链接时或程序运行时转换成物理地址。图 2-6 表示有 A, B 两道程序, 每道程序的逻辑地址都从零开始, 然后由操作系统各自分配到 $a \sim a+i$ 和 $b \sim b+k$ 。

1. 基址寻址

先在基址寄存器中存放本程序的起始地址 (例如如图 2-6 中 A 道程序的起始地址 a), 然后在执行指令时, 由硬件将指令中所表示的地址 (逻辑地址) 和基址值 a 相加, 形成有效地址 (物理地址), 这就是基址寻址原理, 如图 2-7 所示。

另外, 可用基址扩充寻址范围, 如 Intel 8086, 其基址为 16 位地址码, 寻址范围为 64KB, 在执行指令时, 8086 将基址左移 4 位 (即左移 4 位, 补充 0000) 与指令地址相加, 即可得 20 位物理地址, 寻址范围扩大到 1MB, IBM PC 机即是如此。

基址寻址公式: $\text{有效地址} = \text{基址} + \text{逻辑地址}$

2. 变址寻址

变址寻址是将变址寄存器中内容 (变址基值) 与指令中的地址码相加, 形成有效地址, 如图 2-8 所示。图 2-8 所示变址操作对处理一维数组的支持, 用户编写对数组中一个元素进行运算的程序, 然后改变变址寄存器的值 (从 $1 \rightarrow i$), 对程序循环执行 i 次, 就可对数组中 i 个元素逐个处理。也可使变址寄存器中值不变, 而改变指令地址码 (也称变址寻址中的偏移量), 达到同样的目的。如果要处理二维数组, 需用两个变址寄存器。

变址寻址公式: $\text{有效地址} = \text{变址基值} + \text{偏移量}$

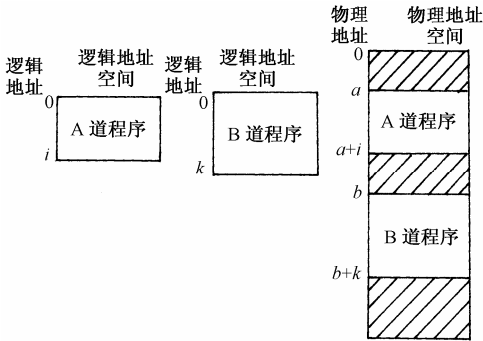


图 2-6 逻辑地址和物理地址空间示意

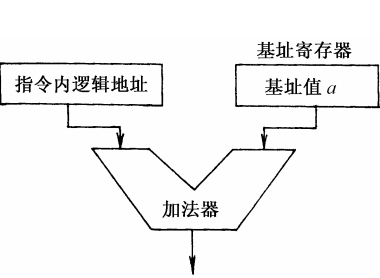


图 2-7 基址寻址

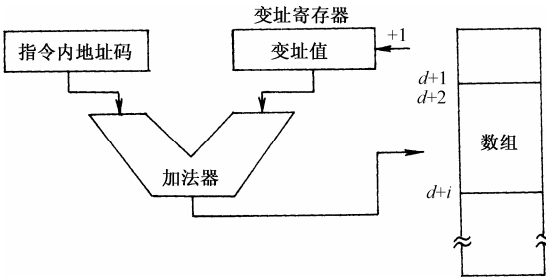


图 2-8 变址寻址

3. 直接寻址和间接寻址

在指令中直接给出操作数的地址, 据此地址即可直接取得操作的数据, 称为直接寻址。

假如按上法取得的不是操作数据, 而是地址, 则称为间接寻址。有的计算机有多级间接方式。例如, 转移指令 “ $\text{JMP } d_1$ ” 和 “ $\text{JMP } (d_1)$ ”, 二者寻址方式不同, 转移结果也不同。前者为直接寻址, 机器执行 “ $\text{JMP } d_1$ ” 指令, 程序转移至 d_1 处继续执行; 后者为间接寻址, 机器执行 “ $\text{JMP } (d_1)$ ” 指令, 若 d_1 单元内容为 d_2 , 则程序转移至 d_2 处继续执行。

4. 相对寻址

指令中的地址码给出的是相对于当前指令（指当前有相对寻址的指令）的位移量 d ，执行本条指令后，程序转移到 $PC+d$ 处。相对寻址时，

$$\text{有效地址} = \text{当前指令 PC} + \text{位移量 } d$$

5. 寄存器寻址

CPU 内有多个内部通用寄存器，这是为减少访问存储器次数、提高运算速度而设置的。若参与运算的数据存放在内部通用寄存器内，对这些寄存器直接寻址，称为寄存器寻址。

6. 立即数寻址

在指令的地址码部分存放的是参加运算的数据。
上述各种寻址方式可以组合使用，例如在一条指令执行过程中实现基址寻址和变址寻址，那么

$$\text{有效地址} = \text{基址寄存器内容} + \text{变址寄存器内容} + \text{指令地址码 } d$$

用户用高级语言编程，不用考虑寻址方式，由编译程序解决。若用汇编语言编程，则应对它有确切的了解，才能编出正确而有效的程序。对于虚拟存储器，由于其逻辑空间比实存空间大许多，用上述方法求得的有效地址仍为逻辑地址，需通过硬件与软件配合才能完成逻辑地址到物理地址的转换。

一台计算机用了哪些寻址方式，需查阅该计算机指令系统说明书，同一种寻址方式在不同的计算机中的实现也会有差别。

2.1.4 指令优化

指令系统是计算机所有指令的集合，程序员用各种语言编写的程序都要翻译成（编译或解释）以指令形式表示的机器语言后才能运行，所以指令系统反映了计算机的基本功能，是硬件设计人员和程序员都能见到的机器的主要属性。早期的计算机指令系统很简单，一些比较复杂的操作由于程序实现，因此计算机的处理速度比较慢。随着硬件价格的下降，指令系统逐步扩充，指令的功能也逐步增强。指令系统的改进是围绕着缩小与高级语言的语义差异以及有利于操作系统的优化而进行的。例如，高级语言中的实数计算是通过浮点运算进行的，对应应在指令系统中设置浮点运算指令能明显地提高速度；另外在高级语言程序中经常用到 IF 语句、DO 语句等，为此在指令系统中设置功能较强的“条件转移”指令是有益的；为了便于实现程序嵌套，设置了 Call 及 Ret 指令……，上述这些措施都是针对高级语言的优化进行的，使生成的目标程序短且运行速度快，同时也便于编译。至于为操作系统的实现或优化而设置的特殊指令，除了各种控制系统状态指令外，还有为多道程序公用数据管理和多处理机系统信息管理用的“测试与置定”、“比较与交换”等指令。

1. 指令格式优化

指令由操作码和地址码组成，上述各种寻址方式主要反映在指令的地址码部分。指令类型不同，地址码的长度要求变化很大，例如操作数据在内部通用寄存器内比它在存储器内，其地址码长度要短得多。为缩小程序代码所占的存储容量，各类指令的长度可以不一致，如在同一个计算机中可以有 1B, 2B, 3B, 4B 等多种长度的指令。从压缩代码的观点出发，希望常用指令的操作码短些，这样使程序的长度也短些。运用哈夫曼（Huffman）压缩的基本概念，可以达到操作码优化的目的。

哈夫曼压缩的基本概念是：出现概率最大的事件用最少的位（或最短的时间）来表示（或处理），而概率较小事件用较多的位（或较长的时间）来表示（或处理），达到平均位数（或时间）缩短的目的。在使用哈夫曼压缩之前，必须先了解每一种指令使用的概率——使用频率 P_i 。如表 2-1 所示，设某计算机 7 条指令（ $I_1 \sim I_7$ ）使用频率 P_i 从 45%~1%。按哈夫曼信息源熵（即平均信息量） $H=-\sum P_i \log_2 P_i$ 公式计算，本例中 $H=1.95$ 。操作码的实际平均长度为 $\sum P_i l_i$ （ l_i 为操作码位数）。信息冗余量为 $1-H/(\sum P_i l_i)$ 。若本例 7 条指令用三位二进制编码表示，则信息冗余量为 $1-1.95/3=35\%$ 。若用图 2-9 所示哈夫曼操作码表示，则信息冗余量为 $1-1.95/1.97=1.015\%$ （ $\sum P_i l_i=1.97$ ）；用扩展哈夫曼操作码表示，则信息冗余量为 $1-1.95/2.2=11.4\%$ （ $\sum P_i l_i=2.2$ ）。哈夫曼操作码表示，冗余最少，但操作码不规整，无法组成指令。扩展哈夫曼操作码表示，冗余增加至 11.4%，但比定长编码（本例用三位）冗余 35%少得多。实用扩展哈夫曼操作码表示用 4—8—12 位等长扩展。

另外，在计算机内存放的指令长度应是字节的整数倍，所以操作码与地址码长度之和应是字节的整数倍，在考虑操作码优化时还应考虑对地址码的要求。故指令格式优化的最终目的是，使用最适当的位数表示指令的操作信息（操作码）和地址信息（地址码），从而取得减小程序存储容量、提高取指速度、简化指令译码网络等效果。

2. 指令系统分析

指令系统从早期的简单形式逐步发展到多种寻址方式、多种数据格式的复杂指令集，是为了提高计算机功能以满足用户日益增长的需求。对已有机器的指令系统进行分析，了解各类指令的实际使用情况，有利于新型计算机的设计。表 2-2 是在 IBM 370 计算机上用不同语言编写的程序所出现的指令频度。该统计数据为机器运行时测得的动态频率，若根据程序代码统计则为静态频率，两者不完全相同，但有一定的关系。

表 2-1 指令操作码哈夫曼优化

指 令	P_i	哈夫曼操作码 OP	l_i	扩展哈夫曼操作码 OP	l_i
I_1	45%	0	1	0 0	2
I_2	30%	1 0	2	0 1	2
I_3	15%	1 1 0	3	1 0	2
I_4	5%	1 1 1 0	4	1 1 0 0	4
I_5	3%	1 1 1 1 0	5	1 1 0 1	4
I_6	1%	1 1 1 1 1 0	6	1 1 1 0	4
I_7	1%	1 1 1 1 1 1	6	1 1 1 1	4

表 2-2 不同高级语言中指令出现的频率

指 令	COBOL	FORTRAN	Pascal
转移	24.2%	18.0%	18.4%
逻辑操作	14.6%	8.1%	9.9%
存取	40.2%	48.7%	54.0%
存储器-存储器传送	12.4%	2.1%	3.8%
整数运算	6.4%	11.0%	7.0%
浮点运算	0.0%	11.9%	6.8%
十进制运算	1.6%	0.0%	0.0%
其他	0.6%	0.2%	0.1%

从分析数据可见，程序的 80%是存取、转移、算逻运算等简单指令，复杂指令的使用仅占 20%。但为复杂指令而设置的微程序却占微程序 ROM 的 80%。复杂指令增加而使 CPU 结构趋于复杂，对编译程序的优化好处并不大。为此提出了精简指令集计算机——RISC(Reduced Instruction Set Computer)，而把传统计算机称为复杂指令集计算机——CISC(Complex Instruction Set Computer)。

RISC 采用 32 位固定长度的指令，指令格式固定，寻址方式少，便于编译优化，减少了硬件控制的复杂性，便于制造。RISC 的 CPU 内有较多的内部通用寄存器，算逻指令的操作数都在寄存器内，访问存储器只有 Load 和 Store 指令，大多数指令能在一个机器周期内完成。RISC 程序代码长度大于 CISC，其原因是：CISC 中的复杂指令在 RISC 中用子程序实现，固定的 32 位指令长度也会使程序代码增加。所以 RISC 程序长度虽有所增加，但增加不多，而每条指令执行的平均周期数却减少很多，因此运算速度可提高几倍。同时，因 CPU 结构简化，可将部分外围电路集成于 CPU 片内，可进一步减少每个周期的时间宽度，更加提高了机器速度。

指令系统从简单发展至复杂主要出发点是：尽量缩短指令与高级语言语义的差距；提高操作系统效率，由此达到提高运算速度的目的。但进一步增强指令功能已不能取得更好效果，反而使硬件结构过于复杂，所以从精简指令入手寻求出路。当然，CISC 仍有其广泛的市场和实用价值，纯 RISC 系统也不理想，所以在今后相当长的时期内，CISC 和 RISC 并重的局面是现实的，客观存在的。如何将 CISC 和 RISC 各自的优点结合起来形成新的体系结构是当前系统结构研究的前沿课题之一。

2.2 总线结构

计算机系统是由许多具有独立功能的模块（如中央处理器——CPU、存储器、输入/输出端口等）互相连接而成的。同时，随着计算机的不断发展和广泛应用，各生产厂商除了向用户提供整套系统外，还设计和提供各种功能的插件模块，让用户根据自己的需要构成自己的应用系统或扩充原有的系统。这些模块间需要互相通信，需要有高速、可靠的信息交换通道。这就产生了总线的概念。简言之，各模块间各种信号线的集合就是总线，如图 2-9 所示。它不仅是互连信号线的逻辑定义，而且还包括这些信号线的工作时序、电气特性、插座信号线排列、信号线个数、几何尺寸、机械特性等一系列指标。

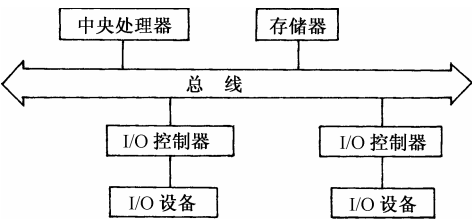


图 2-9 总线连接示意图

有了总线的概念后，使各模块间的信号线可直接互相连接，提高了信号传输的速度，但也产生了互连点繁多、模块间连接凌乱等问题，并且各个厂商纷纷提供各种功能的辅助功能板，信号的种类更多，使总线连接更加复杂紊乱。因此，又进一步提出了总线标准化问题。

总线是信号线的集合，它是组成整个计算机系统各模块的标准信息通路。这些信号线可分成 4 种类型：

- (1) 数据和地址线——决定数据总线的宽度和直接寻址的范围。
- (2) 控制、时序和中断信号线——决定总线功能的强弱，适应性的好坏。对这类信号线

要求控制功能强，时序简单，使用方便。

(3) 电源线——决定使用电源电压的种类，地线的分布及其用法。

(4) 备用线——为用户作性能扩充用，或用户自定义以满足特殊需要。

2.2.1 总线的分类

总线按其功能、规模及接口所处的位置可分成三大类。

(1) 片总线 (Chip Bus)，又称元件级总线 (Component-Level Bus)。它是用微处理器芯片组成的一个很小系统或者构成一块 CPU 插件板所使用的总线。当然也可推广到微处理器芯片内采用的总线结构和单片微计算机所用的总线结构。通常它包括数据总线、地址总线和控制总线三类。

(2) 内总线 (Internal Bus)，又称微计算机总线 (Microcomputer Bus) 或板级总线 (Board-Level Bus) 或系统总线 (System Bus)。它是微型计算机系统内插件间的并行通信总线。如 CPU 模块与存储器模块之间的通信。

(3) 外总线 (External Bus)，又称通信总线 (Communication Bus)。它是系统与系统之间的通信总线。如接口与外部设备的通信，微型计算机与微型计算机之间的通信，等等。随着多处理机、计算机网络的发展，也有为它们设置的外总线标准。

2.2.2 总线结构的特点

(1) 简化硬件设计。由于对信号的引出、电平的极性、逻辑的定义、时序关系等都做了规定，设计者只要按这些标准制作插件板即可，用户只要用符合自己应用系统中总线标准的插件扩充自己的系统即可。所以系统设计者只需着眼于插件板的选择，而不必考虑这些插件板之间的匹配问题，使系统的硬件设计大大简化了。

(2) 系统扩充性好。要在规模上扩充只要插上同类总线标准的插件板即可。要在功能上扩充，也只需按总线标准制造新的功能插件板并插入原系统总线上即可。显然这样使系统扩充既简单容易，又快速可靠。

另外，使用总线结构，用户可以从基本的小系统开始设计，以后随着工作的进展逐步扩充系统的规模和功能，无须改变最初的设计。

(3) 系统更新性能好。可随着新型芯片或模块的不断涌现而更新自己原有的微型计算机系统。一旦新的高性能微处理器、存储芯片或接口芯片出现，只要将这些芯片按总线标准做成各类插件，就能很容易取代原来的插件而更新整个微型计算机系统。

(4) 容易标准化。按照行业公认的总线标准制造标准化的插件板，各厂商不仅能随着芯片技术的发展，制造出总线一致而功能不断更新的新型插件，而且可按统一的总线标准使其系列化。系统设计者只需关注插件板功能而不必关心越来越多的各类芯片的内部细节。

2.2.3 总线通信方式

在系统内各模块之间或系统之间的信息通信过程中，每一时刻只能有一组信息在总线上传输。若有多组信息要传输，只能按顺序分别传输。这样对每一组信息的传输就形成了一个传输周期，而且在这个传输周期里分成 4 个传输阶段，分别完成一定的传输功能。这一切的安排主要是从高速可靠地传输信息这个目的出发的。

(1) 申请分配阶段。一组信息在总线上传输，总有一个要求通过总线进行数据通信的提出者，又有一个被要求进行通信的对象。我们简称提出者为主模块（包括系统），对象为从模块（同样可以是系统）。

当主模块要求在总线上通信时，它首先向总线管理机构——总线仲裁器提出使用总线的申请。总线仲裁器经过判断，认为可以批准主模块使用总线，它就把下一个传输周期的使用权交给主模块。

(2) 寻址阶段。获得总线使用权的主模块要在总线上提出它要进行通信的从模块的“地址”以及进行何种通信的控制信息。当这些信息被从模块接收到时，从模块就会启动，做好相应的通信准备。

(3) 数据交换阶段。这时，主模块与相应的从模块彼此已建立起通信机制，各种信息由发送模块（可以是主模块或从模块）传送到接收模块（可以是主模块或从模块），进行实际的数据交换。

(4) 撤销阶段。一组信息传输完毕，主模块应通知总线仲裁器，并把总线使用权交还总线仲裁器，以便让别的模块能使用总线进行通信。即使刚使用总线的模块需要继续使用总线进行通信，也需要重新向总线仲裁器提出申请。这样才能保证各模块使用总线的机会均等。

为了保证总线通信的高速可靠性，上述4个阶段形成了常用的4种总线通信方式：同步通信方式、异步通信方式、半同步通信方式、分离式通信方式。

1. 同步通信方式

在同步通信方式里，模块之间的通信传输周期是固定的。有精确稳定的系统时钟作为传输周期的“标尺”，通信双方的模块则需严格按时钟标尺进行各自相应的操作。图 2-10 说明了同步通信方式的一般过程。

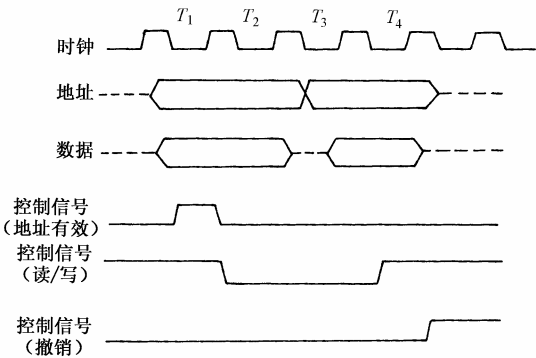


图 2-10 同步通信方式

我们把系统时钟 $T_1 \sim T_4$ 作为一个传输周期。当主模块（如 CPU）准备与从模块（如内存）进行某一通信操作（如读取一个数据）时，主模块首先在传输周期开始的第一个系统时钟 T_1 时，送出准备与之通信的从模块的地址到总线上，并且在地址信号趋于稳定后，送出一个控制信号（地址有效）。凡是挂在总线上的所有模块，只有与这个有效地址符合的模块，即为主模块要求通信的从模块。这个过程就是寻址阶段。

主模块在系统时钟 T_2 送出有关操作控制信号（读/写，这里是读操作）。从模块在操作控制信号的作用下，把主模块所需的数据从相应的存储单元里取到总线上。当然这是需要一

定时间的（存取周期），在系统时钟 T_3 的后半周趋于稳定。主模块在系统时钟 T_4 的前半周把总线上的数据取回主模块内相应的单元（如寄存器），这就是数据交换阶段。

主模块在完成自己所需的操作后，在系统时钟 T_4 的后半周送出一个控制信号（撤销），表明一个传输周期的结束。

这里要说明一点，似乎通信方式第一阶段申请分配阶段没有体现。主要是我们给出的是只有一个主模块的简单系统的例子，就无须申请、分配了，因为总线的使用权始终属于这个模块。如果在诸如 DMA 等系统里，任一模块都可以通过申请被认为是主模块，这时申请分配阶段是不可缺少的。当然总线的仲裁器也是不可少的。

从上面讨论的同步通信方式中我们发现，主、从模块的动作是严格按系统时钟来进行的。比如，从模块在 T_3 的后半周到 T_4 的前半周必须把主模块所需的数据放在总线上，让主模块在 T_4 的前半周取到它的相应单元里。如果这二者不能严格“同步”，则会发生出错。假如从模块需在 T_4 的后半周以后才能送出数据（即存储器速度比较慢），那么在 T_4 的前半周，主模块仍按系统时钟的规定，从总线上取“数据”，而这时总线上的信息并非是从模块送出的数据，仅是总线上的随机信号，主模块因得到随机信息而出错。显然，在同步通信方式里，主、从模块的操作应严格按系统时钟来进行，即所谓速度匹配。由此造成设计者的不便，也缺乏设计的灵活性。

2. 异步通信方式

为了使不同操作速度的模块之间也能进行速度匹配，顺利地进行彼此间的通信，人们提出了异步通信方式。这种方式不再需要主、从模块的操作严格按系统时钟来进行。只是为了主、从模块之间不同速度的配合，增设两条应答信号线（又称握手交互信号线，Handshaking），分别称为请求线和响应线。

我们仍以同步通信方式的例子来说明异步通信方式，如图 2-11 所示。

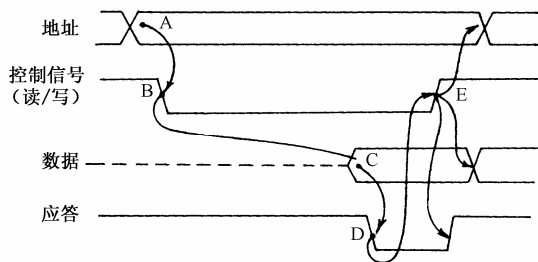


图 2-11 异步通信方式

主模块希望与某一从模块通信（CPU 向存储器读一数据），它把从模块的地址送到总线上（A）。在地址趋于稳定后，主模块发出一个操作控制信号（读信号）（B），这个信号同时作为异步通信方式里的请求信号。只有地址符合的从模块在操作控制信号的作用下进行译码、取数据等一系列内部操作，把数据送到总线上（C）。当数据趋于稳定后，从模块发一应答信号（D）。主模块只有在接收到应答信号后，才认为这时总线上的数据是正确的，于是把数据读到相应的单元里。同时发出撤销信号，这里利用读信号的上升跳变（E）来表示。这个撤销信号使从模块回复应答信号，数据结束和地址有效结束，从而表明一个传输周期的完成。

从异步通信方式可以看到，主模块发出请求后，它一直等到从模块发出应答才可以确认

数据已送出。这段时间可以不定，从而实现不同速度模块间的配合，不必再按系统时钟的“标尺”来进行操作。因此异步通信的关键是，主模块按自己的速度送出请求信号，从模块按自己的速度发出应答信号，从而来掌握彼此匹配的速度。

异步通信方式的最大优点是各种速度的模块都可以很好地配合操作，而各自模块又以自己的最佳速度运行。

3. 半同步通信方式

在上面介绍的两种方式里，同步通信方式简洁，但受系统时钟限制过死；异步通信方式虽灵活，但要增加请求/应答信号线。介于二者之间，提出了另一种通信方式——半同步通信方式，见图 2-12。

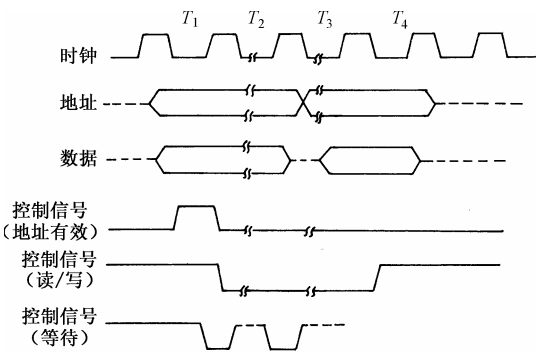


图 2-12 半同步通信方式

半同步通信方式是在同步通信方式里增加一条控制信号线。我们仍以 CPU 读取存储器为例，说明其工作过程。

整个工作过程类似同步通信方式。当主模块要读取从模块数据时，它仍在系统时钟 T_1 时刻发出地址，并确定操作控制信号和地址有效信号。主模块在发送这些信号后，就不断去测试一个“等待”的控制信号，它是由从模块发出的。如果从模块不能按系统时钟 $T_3 \sim T_4$ 间送出数据，从模块就使“等待”信号有效。主模块在 T_2 测出“等待”信号有效，就主动在系统时钟 T_2 后插入一个附加的系统时钟 T_w 。当 T_w 结束时，再测试“等待”有效否，若有效，则继续插入 T_w 。插入 T_w 的个数视“等待”有效的长短而定。只有测试到“等待”无效，主模块才进入系统时钟 T_3 。这也意味着，主、从模块速度的不匹配是依靠“等待”信号来解决的。而“等待”信号的长短，可由从模块根据自己的速度与主模块的速度按系统时钟限制的差异定出。

由此可见，半同步通信方式既像同步通信方式那样有系统时钟一步步的限定，又与异步通信方式里的应答信号一样，只有在“等待”信号结束时才表示从模块的数据就绪。半同步通信方式的优点在于，不同速度的模块能在系统时钟的同步下进行可靠的工作，而无须像异步通信方式那样要设置互锁的请求/应答信号。但半同步通信方式的频率不能太高，否则系统工作会不稳定。

4. 分离式通信方式

同步通信方式和异步通信方式都存在一个问题，当主模块送出地址等信息后，就处于等待状态，从模块被寻址到把有效数据送到总线上这段时间又由从模块的工作速度所决定。这

段等待时间里，总线仍被主模块所占用而又不做任何有效工作。当用户对计算机系统速度越来越“斤斤计较”时，对这段等待时间的“浪费”感到很惋惜，于是提出了分离式通信方式，其本质是异步方式。

这种通信方式的基本思想是：把一个读周期一分为二。当主模块把要寻址的从模块地址等信息送给总线后，主模块就把总线使用权交还总线仲裁器。当然从模块在被启动后，进行自己内部操作。这样，系统总线可以让出来给其他的模块使用。当从模块完成内部操作后，它要变成主模块向总线仲裁器申请占用总线。在允许使用总线后，它要发出原主模块的地址，而原主模块就变成了从模块，并接收原从模块（现主模块）送来的有效数据。由此可见，把寻址阶段作为前一个传输子周期，把数据交换阶段作为后一个传输子周期。

这种分离式通信方式在时间上是节约了，但它的结构却复杂了。每个模块既作为主模块，也作为从模块；每个寻址的模块，都必须把自己的地址告诉对方，以便在下一个传输子周期里能正确寻址，这就增加了信息传输量；总线仲裁器相应也要复杂些。

2.2.4 总线仲裁

在采用总线结构的系统里，当多个模块同时申请使用总线而成为主模块，或在一个模块使用总线期间，另一个模块要求使用总线来完成比现使用总线模块更加紧迫的任务时，就存在一个总线究竟给谁使用的问题。它类似于中断请求的裁决问题，因此也必须有一个总线仲裁器来对总线申请做出裁决。

在只有一个主模块（CPU）的简单微型计算机系统里，不存在主模块竞争问题。在有 DMA 等操作的微型计算机系统、多处理器系统、网络等系统里，几乎随时都可能发生各模块对总线的争用问题。在这些场合，总线仲裁能力是不可少的。所以在近年出现的总线标准里（如 VME）都有总线仲裁能力的标准。

总线仲裁的原理类似于中断处理，它们的仲裁方法也类似，主要有串行总线仲裁和并行总线仲裁。

1. 串行总线仲裁

根据各模块任务的轻重缓急，赋予它们各自的优先权。总线仲裁的基础是判断各申请使用总线的模块优先权的高低。

串行总线仲裁的优先权排列是按链接的位置决定的。每个模块只有在它前面的模块（即优先权高）没有使用总线时，它才能申请使用总线。换句话说，当某个模块要申请使用总线时，它必须明确在它之前的模块均未使用总线。一旦总线仲裁器把总线使用权交给它，一方面它获得使用权，另一方面要通知它后面的模块（优先权低的）不能申请使用总线。这个信息是依靠串行链逐个传递下去的，所以称串行总线仲裁。

如果一个优先权较低的模块正在使用总线，此时一个优先权较高的模块申请使用总线，按理能立即“中断”优先权较低的那个模块的总线操作，类似中断结构里的“嵌套”过程。但这往往会造成差错，为此在总线仲裁里要增加控制信号，来避免此刻中断总线操作。只有当这个模块的总线操作完成，并立即把总线使用权交还总线仲裁器时，再由总线仲裁器把总线使用权交给那个正在申请的优先权较高的模块，如图 2-13 所示。

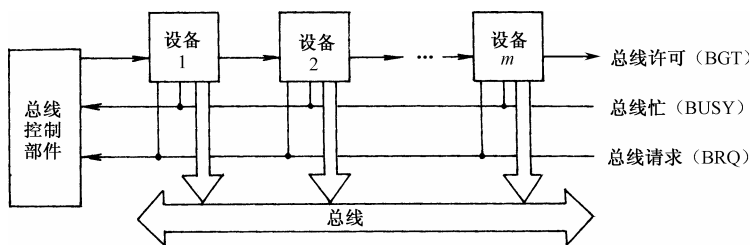


图 2-13 总线结构的串行仲裁

优先权的赋予有两种方式。① 固定式优先权是指由系统设计者根据系统功能事先按处理事务的轻重缓急，赋予每个模块固定的优先权。这种安排使各模块在运行过程中，其优先权不会改变。② 转换式优先权，是指按某一转换规律进行优先权的重新赋予。它使各模块的优先机会均等。例如起始时按一定次序安排了各模块优先权，凡申请使用过总线的模块，在其完成总线操作后，其优先权变为最低（即排到队尾）。或在一次总线操作后，按某个序列变化。这些方法已在 MULTIBUS 等多种总线里实行。

2. 并行总线仲裁

在并行总线仲裁系统中，实际上是利用外部逻辑的硬件编码来进行模块的判优问题。外部逻辑构成一个并行优先权网络，每个模块按优先权顺序把申请信号接到外部逻辑上。这样每个模块都是并行地接入外部逻辑。当有模块申请使用总线时，它把申请信号提交给外部逻辑。该外部逻辑由优先权编码器和译码器组成，经判断是当前申请使用总线的模块中最高优先权的模块，总线仲裁器就通知该模块使用总线。当其他模块提出申请时，若经外部逻辑优先权网络比较下来，其优先权低则被拒绝。只有高于现行模块优先权的模块才可能被接受。如图 2-14 所示。

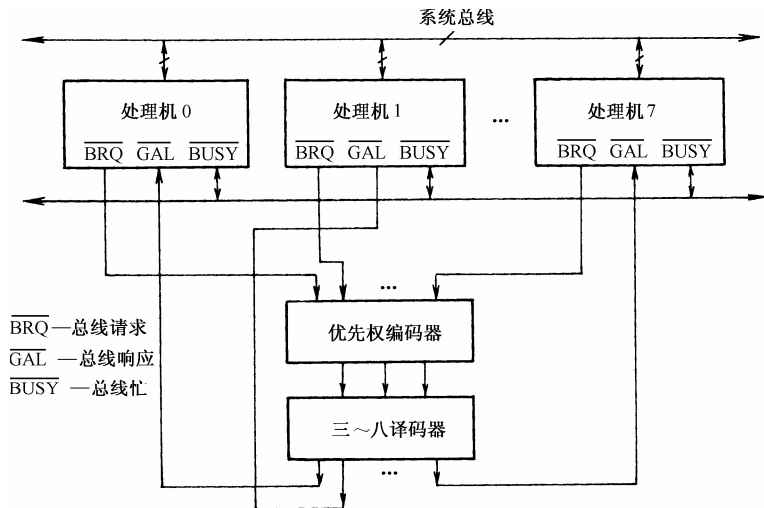


图 2-14 总线结构的并行仲裁

并行总线仲裁明显的优点是，要改变优先权只要改变接入外部逻辑的申请信号次序即可。

2.2.5 总线标准

采用总线结构有许多方便之处,为使人们在组成各种模块系统时更加方便,推出了由国际组织正式公布或推荐的互连各个模块的总线标准。总线标准为各模块互连提供了一个标准的接口界面,各类用户不必考虑对方模块的接口方式,只需根据总线标准的要求来实现接口功能。因此极大地方便了用户的软硬件设计。

在总线标准里,必须明确总线中各信号线的定义、逻辑关系、时序要求、信号表示方法、电路驱动和抗干扰能力等。其他如导线的物理特性、信号线在插座上的位置次序等也要规定清楚。总之,必须有一个统一的标准(或叫“总线规范”)便于各个厂商遵循。因此任何一个厂商生产的各种模块,只要符合标准,都能插入总线槽,和系统总线接通,协调工作。

经过多年的竞争,一些总线在广泛的使用中逐步形成事实上的总线标准。美国电气与电子工程师协会(IEEE)等权威性组织又进行了专门的评选并编号,提出了适合不同应用水平的几种总线标准。这些措施使总线标准成为计算机系统不可缺少的一个组成部分。

1. MULTIBUS总线

MULTIBUS 即 MBI 总线(IEEE-796 总线标准)是 Intel 公司为其 16 位微型计算机系统设计的一种总线标准。MULTIBUS II (MB II) 是 Intel 公司为 32 位机设计的总线标准。由于 Intel 公司生产了供这种总线接口使用的整套系列通用接口集成电路,装配成大量符合这种总线标准的各种功能模块,使系统设计者使用非常方便,因此 MULTIBUS 总线成为世界上使用最普遍的微型计算机总线。

MULTIBUS 总线结构是根据主从概念构成的,包括在系统中控制 MULTIBUS 接口的主设备和从设备,其中从设备根据主设备提供的设备码地址和命令而动作。在主、从设备之间交换信息时,允许不同速度的模块使用 MULTIBUS 接口。

MULTIBUS 总线的另一个特点是,在多处理机系统中能够连接多个主模块。该 MULTIBUS 接口可按菊花链串行式优先权方式或并行优先权方式来提供连接多主模块的控制信号。采用并行优先权方式可使 16 个主模块共享总线资源。

2. VME总线

VME 总线(IEEE-1014 总线标准)是由美国 Motorola 公司研制的 VERSABUS 经过改进后的一种 32 位总线标准。由于 Motorola 公司将 VME 总线首先用于 M68000 微处理器组成的微型计算机系统,并且生产大量该系列的接口集成电路,使得 VME 总线受到各方面的重视及广泛使用,不少微机系统和工作站都采用它。

VME 总线主要分成 8 个功能模块和 4 条子总线,8 个功能模块是:

(1) 系统时钟驱动器。它提供 16MHz 的系统时钟。其他模块可由此时钟导出自己的内部定时信号。

(2) 功率模块。提供系统复位信号和电源掉电信号。

(3) 数据传送主设备。保证总线经仲裁确认的主设备的总线使用权。

(4) 数据传送从设备。对地址线及主设备提供的选通信号进行译码,作为从设备的响应。

(5) 总线请求器。它是一组控制逻辑,是总线仲裁机构的一部分,用于申请控制数据传送总线。

(6) 总线裁决器。由它对申请控制数据传送总线进行仲裁。由于 VME 总线定义单级、固定优先权和轮转式,用户可以进行事先的选择。

(7) 中断器。它设置各模块可能提出的中断请求。可以是菊花链串行式或外部逻辑并行式。

(8) 中断处理器。它在裁决中断请求优先级后，获取对总线的控制。

4 条子总线各自完成一个具体功能，它们是：

(1) 数据传送总线。包括了数据总线、地址总线、控制总线。

(2) 裁决总线。它实现各主模块对系统总线申请使用的裁决。它是整个 VME 总线仲裁机构的一部分。它产生 5 个控制信号来完成总线使用的裁决。

(3) 中断总线。它实现各模块对中断申请的裁决。它产生 4 个中断处理信号。

(4) 公用总线。它提供整个系统支持其他设备所需的信号线以及系统复位、电源等信号线。

3. PCI总线

随着 Windows 图形用户界面和多媒体技术广泛应用，要求系统具有高速图形处理和 I/O 吞吐能力，这使原有 PC 机上的 ISA, EISA, VESA 总线不能适应新的发展及需求，逐渐成为系统速度提升的瓶颈。同时，各厂商为使自己机器上的总线成为工业标准而进行激烈的竞争。为了统一总线标准，使之成为 21 世纪公众广泛遵守的工业标准，也为了已有总线能相互兼容，逐步过渡到统一的标准，为此在 1991 年下半年，Intel 公司首先提出了 PCI 的概念，并联合 IBM, HP, Compaq, Apple, Motorola, NEC, 富士通, Acer, Philips, Siments, NCR, SUN, MS 等 100 多家大公司商讨，成立了以 Intel 公司为首的 PCI 集团。全名是 Peripheral Component Interconnect Special Interest Group（外围部件互连专业集团），简称 PCISIG。PCI 是一种先进的局部总线，广泛应用于当前的 PC 机、工作站、便携式微机，是 21 世纪的系统总线标准。

PCI 总线标准的特点是：

(1) 数据传输速率高。32 位数据传输速率为 133MB/s，64 位数据传输速率可达 532MB/s。它大大缓解了数据 I/O 瓶颈，使高性能 CPU 功能得以充分发挥，适应高速设备数据传输的需要。

(2) 多总线共存。采用 PCI 总线的计算机系统可以使多种总线共存，容纳不同速度的设备一起工作，构成分层次的多总线系统，如图 2-15 所示。其关键是 Intel, IBM, DEC 等公司提供的一整套的 PCI 与 CPU、PCI 与 ISA/EISA、PCI 与 MCA 等桥接芯片组，实现了多种总线的兼容，扩大了系统兼容性。

(3) 独立于 CPU。PCI 不依附于某一具体处理器，即 PCI 可支持多种处理器和未来新的处理器，在更新处理器时，更换相应的桥接组件即可。

(4) 自动识别与配置外设。有即插即用（Plug and Play）功能，用户使用方便。

(5) 并行操作能力。

PCI 总线的主要性能如下：

(1) 总线时钟频率为 33.3MHz/66.6MHz。

(2) 总线数据宽度是 32 位/64 位。

(3) 最大数据传输速率是 133MB/s（532MB/s）。

(4) 支持 64 位地址寻址。

(5) 适应 5V 和 3.3V 电源。

著名总线还有：IBM 公司的 MCA（微通道总线）、STD 总线（用于工业控制计算机）、Future BUS（由 IEEE 主持，VME 联盟、Multibus 集团、美国海军及各大学专家参与下开发的，面向未来的真正开放的总线标准）等。现列于表 2-3 中供参阅。

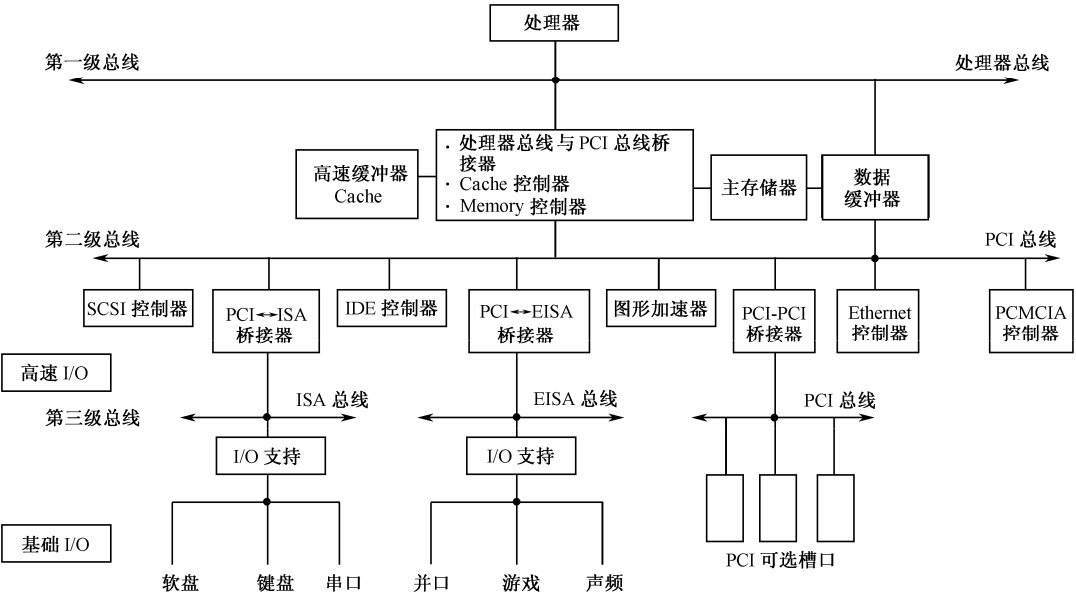


图 2-15 PCI 总线层次结构

表 2-3 流行总线性能一览表

总线名称	发布年代	IEEE 编号	适用机型	AB 宽度	DB 宽度	通信方式	数据传输速率 (MB/s)	BUS CLK (MHz)	仲裁方式	多路复用	引脚数目	负载能力	驱动技术	64 位扩展	并发工作	自动配置	多机系统
S-100	1975	696	8085 Z80	20	8	同步	2	2	集中	非	100	8	TTL	不可			
STD	1978	961	Z80 8088	20	8	同步	2	2	集中	非		无限制	TTL	不可			
MBI	1976	796	8086	24	16	同步				多路			TTL				可
PC-XTBUS	1982		8088	20	8	同步	4	4	集中	非	62	8	TTL	不可		无	
ISA(PCATBUS)	1985	P996	286	24	16	同步	16	8	集中	非	62+36	8	TTL	不可		无	可
H1			日立 WS	32	32		133	20~33.3			137	7		无规定	可	可	
VME	1981	P1014	680x0	16/32	16/32	异步	40	10		非	96/128		TTL				
MBII	1983	P1296	80x86	32	16/32	同步	40	10		多路	96		TTL				可
EISA	1988		386, 486 P5	32	16/32	同步	33	8.33	集中	非	198	6	TTL	无规定			可
VESA	1991		486	32	32	同步	266	66	集中	非	90	6	TTL	可	可		
MCA	1987		PS/2	32	32/64	异步	40/171	10			109/178	无限制	TTL	可			可
PCI	1991		P5 Power PC	32/64	32/64	同步	133/266	33 66		多路	53/92	3	TTL	可	可	可	可
Future BUS+	1991	P896		32/64	32/64 128/256	异步/同步	400~3 200	100		多路	192		BIL	可			可
S-BUS	1990			32	16/32	同步	57	14~28			96		CMOS				

4. Future BUS+

Future BUS+标准是在 VME 国际贸易协会、Multibus 制造商集团、美国海军下一代计算

机资源计划委员会、IEEE 微处理机标准委员会以及来自各公司和大学的专家、学者协作下开发的。其目的是要开发一种真正开放的总线标准，能支持 64 位地址空间和下一代多处理机所要求的吞吐率，并且必须可以向上延伸或可扩展，与特定的结构以及处理机技术无关。

Future BUS+标准的特点是：

- (1) 与结构、处理机技术无关，对所有设计者可用。
- (2) 采用全异步、握手控制的数据传送定时协议。
- (3) 对于高速数据块传送，采用可选的“源-同步（包）”协议。
- (4) 全分布的总线并行仲裁协议支持多种总线通信业务，包括广播（一个发送方、多个接收方）、广集（多个发送方、一个接收方）及三方通信业务。
- (5) 支持高可靠性和容错应用，对带电插拔部件（卡）有保护措施。所有线上都带奇偶校验或反馈校验，在模块失效时，由于不用菊花链式信号，能方便地对系统进行动态重构。
- (6) 使用多级机制锁定模块，避免死锁或活锁。
- (7) 采用线路交换和分离协议，支持实现远程锁定与类似 SIMD（单指令多数据）操作的存储器命令。
- (8) 支持带多优先级级别以及一致性优先级处理的实时任务的计算，支持分布式时钟同步协议。
- (9) 数据总线宽度动态可变（从 32 位到 64，128，256 位），支持 32 位或 64 位寻址，以满足不同带宽的需要。
- (10) 通过多总线互连大型系统的递归协议，直接支持基于监听协议的、有高速缓存的多处理机。
- (11) 使消息传递协议与多计算机的连接、应用和接口设计相兼容。

Future BUS+ 标准信号线定义见表 2-4。

表 2-4 Future BUS+ 标准信号线

类 别	名 称	位 数	功 能
信息线	地址数据复用线	64	64 位地址和低位 64 位数据复用
	高位数据线	64~192	可选用，组成 128，256 位数据线
	标志线	8	可选用，扩充寻址/数据方式
	状态线	8	从方响应主方命令的回答信号
	权力线	3	用于特殊的总线业务
	奇偶线	6~34	每个字节宽的线中配一条奇偶线
应答线	地址握手线	3	传送地址时，主方和从方应答联络
	数据握手线	3	传送数据时，主方和从方应答联络
	总线传输控制线	1	协调总线控制
仲裁线	总线仲裁线	8	传送总线竞争者优先级编号
	同步仲裁线	3	协调握手信号
	条件仲裁线	2	用于特殊情况仲裁
	仲裁奇偶线	若干	用于仲裁线的奇偶校验
	中央仲裁线	4	由中央仲裁器使用
其他	地理地址线	5	传送地理地址（卡、槽地址）
	复位线	若干	总线初始化

对应于 32, 64, 128, 256 位的 Future BUS+数据总线宽度, 该总线标准的总位数分别是 91, 127, 199, 343, 为了与时钟、电源及公用设备连接, 还需要一些附加线。

Future BUS+标准之所以能达到技术独立的目的, 是因为它建立的是在基本协议与物理原理基础上的协议, 并在优化后能获得很高的通信效率, 而不是依赖于处理机的类型或升级换代。定时和握手协议由操作上的规则支配, 而不受设备延迟和俘获窗口等技术因素的限制。广播方式可使速度不同、结构不同的模块方便地连接到 Future BUS+上, 只要物理的逻辑电平能符合 Future BUS+传送信号的要求, 则标准规范可以用任何系列的逻辑 (TTL, BTL, COMS, ECL, GaAs 等) 来实现。结构独立性可为高速缓存一致性提供灵活通用的解决方法, 使其他高速缓存协议与之兼容。同时, 也为多计算机环境下所用的消息传递协议提供良好的支持。因此, 可以按 Future BUS+层次方式用一些较小的子系统构成大型系统。可以采用分离、全互锁、全递归、分层高速缓存等方式, 使用 VLSI 总线接口来实现。Future BUS+的结构独立性增加了构建多处理机系统的灵活性。

5. USB

USB (Universal Serial Bus) 是当前许多计算机采用的通用串行总线。USB 以 Intel 公司为主, 并有 Compaq, IBM, DEC, NEC, Microsoft 等公司参与共同开发, 于 1996 年 2 月发布 USB 1.0 版本, 目前已发展到 USB 3.0 版本。1998 年后, 随着在 Windows 98 中内置 USB 接口的支持模块, 加上 USB 设备日益增多, USB 得到了广泛应用。

USB 有 4 根线, 其中 D+和 D-是一对双绞线, 传送信号, VBus 和 GND 是电源线, 电源电压为 4.75~5.25V, 最大电流为 500mA。USB 设备的电源供给有两种方式: 自给方式 (设备自带电源) 和总线供给方式。USB 对设备提供的电源是有限的, 当 USB 设备第一次被 USB 主机检测到时, 设备吸入的电流值应小于 100mA。

USB 系统拓扑结构采用星形层式结构, 如图 2-16 所示。

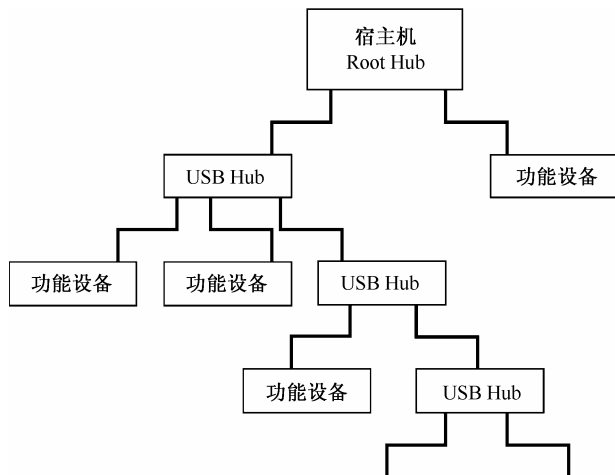


图 2-16 USB 的拓扑结构

这种结构是星形层层向上级联方式, 最多允许连接 127 个设备, 最上层是 USB 主控器。由于 USB 不采用存储转发技术, 所以不会对下层设备引起延迟。宿主机是一个带有 USB 主控制器的 PC 机, 在 USB 系统中只有一个主机, 它有根 Hub (Root Hub), 分别完成对传输的初始化和设备的接入, 主机控制器负责产生由主机软件调度的传输, 然后再传给根 Hub。

USB 的系统软件支持包括：

- (1) USB 设备驱动程序 (USB Device Drivers)。通过 I/O 请求包发出给设备的请求，完成对目标设备的设置。
- (2) USB 驱动程序 (USB Drivers)。在设备设置时读取 USB 设备特征，并根据特征，在请求发生时，组织数据传输。
- (3) 主控制器驱动程序 (Host Controller Driver)。完成对 USB 交换的调度，并通过根 Hub 或其他 Hub 完成对交换的初始化。

USB 有如下 4 种基本传输类型：

- (1) 控制传输。控制 (Control) 传输是双向的，有 Setup 阶段、Data 阶段和 Status 阶段，其中 Data 阶段可有可无。USB 设备必须用端点 0 (即设备内 0 号寄存器) 完成控制传送。实际传输过程为：总线空闲状态→主机发设置 (Setup) 标志→主机传送数据→端点返回 (Status) 成功信息→总线空闲状态。
- (2) 批传输。批 (Bulk) 传输可以是单向或双向的，用于传送数据块，时间性要求不高，但要确保正确，出现错误则重传。典型应用是扫描仪、打印机、静态图片输入。实际传输过程为：当端点处于可用状态，且主机接收数据时，总线空闲状态→发送 IN 标志→端点发送数据→主机返回成功收到→总线空闲状态。若主机发送数据时，总线空闲状态→发送 OUT 标志→主机发送数据→端点返回成功→总线空闲状态。当数据出错时，接收方则不是返回成功，而是请求重发。
- (3) 中断传输。中断 (Interrupt) 是单向的，仅输入到主机，用于不固定的少量的数据传输。USB 中断采用查询方式，查询周期为 1~255ms，即最快的查询频率为 1kHz。在信息传输过程中如果出错，则需在下一个查询中重传。典型应用是键盘、鼠标。实际传输过程为：当端点处于可用状态时，总线空闲状态→主机发 IN 标志→端点发送数据→主机返回成功→总线空闲状态。
- (4) 等时传输。等时 (Isochronous) (同步) 传输可以是单向或双向的，用于传送连续的、实时性强的数据，要求传输速率恒定，忽略传输中错误，即数据传输出错也不重传。传送最大数据包是 1024B/ms。典型应用是音频设备、数字音响、数码相机。实际传输过程为：总线空闲状态→主机发 IN (或 OUT) 标志→端点 (或主机) 发送数据→总线空闲状态。

USB 以包的形式传输，包是组成 USB 交换的基本单位，类似于帧。USB 总线上的每一次交换至少需要三个包才能完成：标志包、数据包、握手包。

(1) 标志包。USB 是一种基于标志 (Token) 的总线协议，所有交换以标志包为起始。标志包 (也称令牌包) 定义要传输的类型。标志包格式如下：

SYNC	PID	ADDR	ENDP	CRC
------	-----	------	------	-----

SYNC——同步符，8 位。

PID——包类型域，8 位，包类型有 4 大类共 9 种，见表 2-5。8 位 PID 中高 4 位用于包的分类编码，低 4 位用于校验。

ADDR——设备地址域，7 位，共有 128 个地址。确定包的传输目的地。

ENDP——端点域，4 位，一个设备可有 16 个端点号 (寄存器号)。确定包传输到设备的哪个端点。

CRC——循环冗余校验域，5 位。用于 ADDR 和 ENDP 校验。

表 2-5 PID 包的类型

包类型	PID 名称	PID ₃	PID ₂	PID ₁	PID ₀	说 明
Token	OUT	0	0	0	1	发送包
Token	IN	1	0	0	1	接收包
Token	SOF	0	1	0	1	帧开始包
Token	SETUP	1	1	0	1	设置包
Data	DATA ₀	0	0	1	1	数据包
Data	DATA ₁	1	0	1	1	数据包
Handshake	ACK	0	0	1	0	应答包
Handshake	NAK	1	0	1	0	无应答包
Handshake	STALL	1	1	1	0	挂起包
Special	PRE	1	1	0	0	预告包

标志包内的 SOF 是帧开始包。主机在每一帧开始时广播到所有全速设备，每隔 1~1.05ms 广播一次。SOF 只能在标志包中，数据包、握手包不能与 SOF 放在一起。SOF 包格式如下：

SYNC	PID	FRAME NUMBER	CRC
------	-----	--------------	-----

FRAME NUMBER——帧号，11 位。

系统软件从设备中读取信息时，使用接收包（IN），接收交换中有 4 种传输类型，即中断传输、批传输、控制传输、等时传输。系统软件需要将数据送到设备时，使用发送包（OUT），发送交换中有三种传输类型，即批传输、控制传输、等时传输。在控制传输开始，主机发设包（SETUP），传送主机请求让目标设备完成。

（2）数据包。数据包格式如下：

SYNC	PID	DATA	CRC
------	-----	------	-----

DATA——数据域，0~1023 位。由 PID 指明 DATA₀，DATA₁ 两种类型数据包。DATA₀ 用于设置包内。

（3）握手包。由数据接收方发送至发送方，其格式如下：

SYNC	PID
------	-----

握手包有三种类型：应答包（ACK）表示接收数据正确；无应答包（NAK）表示功能设备不接收来自主机的数据，或者没有数据返回给主机，也可通知根 Hub 和主控设备，无法返回数据；挂起包（STALL）表示功能设备无法完成数据传输，并需主机解决故障，使设备从挂起状态恢复正常。

（4）预告包。当主机在低速方式下与低速设备通信时，主机以预告包作为开始包，然后与低速设备通信。其格式与握手包类似。低速设备只能支持控制传输和中断传输，且通信包内仅限 8 字节的数据。

USB 技术的应用是计算机外设总线的重大发展，它之所以能得到广泛支持和迅速普及，是因为它具有众多特点：

（1）用一种连接器类型可连接多种外设。USB 对连接外设没有任何种类限制，仅提出了准则和带宽上界，使用统一的 4 针插头，实现了计算机常规 I/O 设备、部分多媒体设备、通信设备及家用电器统一为一种接口的愿望。

（2）使用一个接口连接大量外设。USB 采用星形层式结构和集线器（Hub）技术，允许

一个主机连接 127 个外设，两个外设间电缆长度可达 5m，扩展灵活。

(3) 连接简单快速。USB 能自动识别 USB 系统中设备的接入和拆走，真正做到即插即用，且是热插拔连接。

(4) 总线提供电源。USB 能提供+5V，500mA 的电源，供低功耗设备（如键盘、鼠标、Modem 等）使用，同时采用 APM（Advanced Power Management）技术，节省能源。

(5) 速度快。USB 1.1 允许传输速率达到 1.5Mb/s，USB 2.0 允许速率高达 480Mb/s。

USB 也存在一些问题，尽管理论上可允许多层连接 127 个外设，但实用中没有这么多；虽然可提供 500mA，但遇到高功耗设备，会导致供电不足。

USB 已在 PC 机的多种外设上得到应用，如扫描仪、数码相机、数码摄像机、音响、显示器、软驱、网卡、Modem、打印机、键盘、鼠标、游戏杆等，范围十分广泛。还出现了 USB 转接设备，提供 USB 到 SCSI、USB 到 PCI 的转换，使其他非 USB 接口的外设可接到 USB 上使用。对于笔记本型移动计算机，使用 USB，方便外设连接，结构简化、散热改善，促进更高主频的处理器应用到移动计算机中去。

2008 年推出的 USB 3.0 标准，其传输速率达到 4.8Gb/s，是 USB 2.0 的 10 倍，USB 3.0 标准又被称为“PCI Express over cable”，它的特点是：支持通用 I/O 接口，降低能耗；改善计算机、消费产品和移动产品领域的协议效率；支持快速同步移动能力；支持光学和数字组件规范；具有传统 USB 技术易用性和即插即用的特性；与 USB 2.0 兼容。USB 3.0 接插件分为 A、B 两种插头和 B、AB 两种插座，AB 插座可兼容 A、B 两种插头。USB 3.0 插头的针脚有 9 针，其中 4 个针脚和 USB 2.0 的形状、定义完全相同，而另外 5 根是专用于 USB 3.0 的。USB 3.0 可使用铜缆和光纤，使用光纤后的传输速度可达到 USB 2.0 的 20~30 倍。

IEEE 1394 是高性能串行总线标准，提供点对点传输功能，同时支持同步和异步传输模式，可以连接 63 个设备，可以同时传输数字视频及数字音频信号，采集和回录过程中没有信号损失。因此，IEEE 1394 更适合多媒体设备。IEEE 1394 最大传输速率为 3.2Gb/s，不及 USB 3.0。同时它的普及度也不及 USB 3.0。因此，计算机系统或外部设备往往同时拥用 USB 接口和 IEEE 1394 接口。

USB 3.0 技术得到了 Intel，HP，NEC，NXP，Texas Instruments 等公司的支持和推广，应用于个人计算机、消费和移动类产品的快速同步即时传输，前景甚优。

2.3 存储系统概述

现代计算机系统主要由下列硬部件构成：中央处理机（CPU）、存储系统（主存、辅存）、总线结构、I/O 通道、外围设备、电源等。存储系统占据重要地位。多年来，存储系统的容量数量级和性能变化很大，但三项基本要求——大容量、高速度和低成本却始终未变。促成这种发展的主要因素，可概括为下面两条规则：

(1) 扩展计算机能力的规则。使中央处理机所具有的执行能力得到最大限度的扩展。

(2) 扩展存储器的规则。使所具备的存储容量得到最大限度的扩展。

这两条规则的相互关系是错综复杂的。在某些情况下，由于科学、工程和数学问题的复杂性，提出了对中央处理机能力的更高要求，又由于数据处理和以数据为主的应用中出现了庞大的数据存储量，则提出了对大容量存储器的迫切需要。由于这两条规则的影响，造成了

中央处理机和主存储器之间在速度上的差距不断增大,正是为了弥补这种速度上的很大差距,才采用了存储器按模交叉存取,以及诸如高速缓冲存储器、虚拟存储器、并行存储器和存储器分级结构等许多新的结构概念,使主存储器演变发展成了存储系统。

2.3.1 存储器容量、速度与价格的关系

容量、速度和价格是评价存储器性能的主要依据。存储器容量用下式表示:

$$S_M = Wlm$$

式中, W 为存储器字长,用位数 (bit) 或字节数 (Byte) 表示; l 为存储器字数; m 为并行工作的存储器个数。

存储器的速度可用访问时间 T_A 、存储周期 T_M 或频宽 B_m 表示。 T_M 一般比 T_A 大。 T_A 是 CPU 启动一个访存操作后必须等待的时间,在确定 CPU 与存储器的时间关系中是重要的。在 $(T_M - T_A)$ 余下的时间里, CPU 和存储器可以同时进行各自的操作。 T_M 是存储器进行一次存取操作的时间, B_m 是存储器被连续访问时,单位时间内传送的数据量,通常用每秒传送信息的位数或字节数表示, $B_m = W/T_M$ 。若数据总线宽度 w 与存储器字长 W 不一致,则 $B_m = w/T_M$ (b/s)。若有 m 个存储器并行工作,则总的存储器频宽为 $B_m = wm/T_M$ 。 B_m 为存储器可能达到的最大频宽。实际工作时,存储器不会一直连续被访问,因此其实际频宽往往比 B_m 小。

存储器的每位价格可表示为

$$c = C/S_M$$

式中, C 是具有 S_M 位存储容量的存储器总价格。

每位价格 c 不仅包含了存储单元本身的价格,也包含了存储器外围电路的价格。存储器总价格 C 正比于 S_M/T_M (或 S_M/T_A),即正比于 Wlm/T_M (或 Wlm/T_A)。

设计存储系统的主要目标是:在尽可能低的价格下,提供尽可能高的速度及尽可能大的存储容量。

由于能与处理机速度相匹配的高速存储器的价格太高,因此总容量要求很大的存储系统,仅采用单一工艺的存储器是行不通的,机器必须有多种不同的存储器,使所存信息按各种方式分布在物理特性不同的存储部件上。它至少需要两种:主存和辅存。主存中存储当前正被 CPU 使用的程序和数据,辅存中存储 CPU 暂不使用的 (或称待命的) 程序和数据,主存和辅存中的信息调度由操作系统中的存储管理进行。

早期, CPU 与主存在速度上是相匹配的。但是随着半导体工艺的发展,虽然主存速度有了显著的提高,然而同期内 CPU 的速度提高得更快,所以必须从系统结构上采取措施,解决这个速度差,否则 CPU 的高速效能发挥不了。

从容量上分析,尽管主存容量正在快速地增长,然而由于系统软件和应用软件的迅速发展,主存不能满足存放整个系统程序和用户程序的要求,只能储存必须驻留主存的操作系统的相关部分,然后按需调进主存的操作系统部分模块,以及当前正在使用的支撑软件工具 (如编译程序、编辑程序、数据库管理系统等) 和用户程序中当前正在执行的部分。由于主存容量增长总是落后于无原则存入主存的信息量的增长,即使主存容量达到上千兆字节时,情况仍然如此。因此,计算机系统中同时具备主存和辅存的状况会长期存在。

由于价格 C 正比于 Wlm/T_M ,速度越高,容量越大,则价格越高,它们之间是互相矛盾的,无论主存技术还是辅存技术都是如此。存储技术的多样性以及存储系统的形成,主要原

因就在于此。就一种存储器分析，高速度只能以高成本为代价，这是内部固有特性。因此需要在价格、速度、容量之间进行折中。其中， l 和 T_M 主要与器件工艺有关，而 W 和 m 则由系统结构确定。

综上所述，计算机系统和应用的发展不断对存储系统提出更高速和更大容量的要求，但受到价格的制约，虽然半导体工艺（尤其是存储芯片的集成度）不断发展，成本显著下降，但要取得良好的性能价格比，还得从系统结构上改进，采用存储体系，软、硬结合，达到优化目的。

2.3.2 存储系统的层次结构

在合理价格的前提下，高速、大容量不能只用单一种类器件来实现，因此从一开始计算机内部就至少有两种存储器——主存和辅存。这是否说明，只要计算机内有多种存储器就构成了存储体系（或称存储层次，Memory Hierarchy）呢？其实不是，光有多种存储器，而它们之间如果不能形成有机的整体，只能说有了存储器系统而并无存储体系。

早期的计算机辅存（也称外存）是视作外设的一部分，其编址与主存（也称内存）的编址无关，CPU 一般直接与内存联系，外存的信息先要经过 CPU 的 I/O 操作调至内存，然后 CPU 才能执行。而且，程序在外存中分块的位置和调入内存的地址，以及程序如何一块一块地由外存调入内存的调度过程，均由程序设计者确定。所有这些，计算机系统均不能自动地进行。因此主存和辅存并未通过系统结构和操作系统有机地联系在一起，两者不构成一个整体，从而也不存在存储体系。其后，随着通道与 I/O 处理机的出现，使得内存、外设、CPU 并行工作，一块程序的执行和另一块程序的调进/调出可同时进行，以提高机器的吞吐量。由于主、辅存数据传输速率差别很大，当辅存通过通道与主存交换信息时，主存仍有许多空闲时间，这样在等待辅存内部操作的时间内，CPU 可插空访问主存。其时间关系如图 2-17 所示。CPU 与辅存交替使用主存，即分时使用主存，在客观上实现 CPU 和辅存并行工作。这样主存速度往往成为计算机系统提高效率的瓶颈。

多道程序的出现及发展，促进了管理程序向操作系统演变和发展，还逐步形成了支持主、辅存之间调度定位的“辅助软、硬件”。在系统结构上，通过软硬结合把主存和辅存统一成一个整体，从整体观察，其速度接近于最快、最贵的主存，容量却是辅存的，而每位平均价格接近于廉价慢速的辅存。只有做到这一点，才算形成了存储层次（体系），如图 2-18 所示。

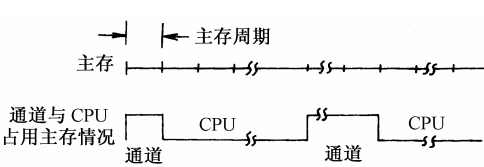


图 2-17 CPU 与通道分时使用主存

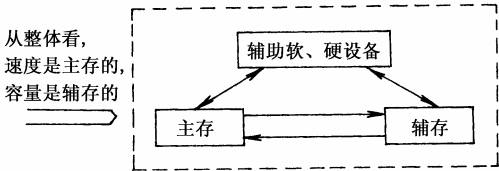


图 2-18 “主-辅”存储层次

存储层次的形成，提高了存储器系统的性能价格比，以接近于辅存的每位价格构成大容量（等于辅存容量）的快速主存，这当然是我们所希望的。这样的体系不断地发展和完善，就逐步地形成了广泛使用的虚拟存储系统。面对虚拟存储系统，程序员可在该虚拟空间内，以统一编址形式编程。我们把这种指令地址称为虚地址（虚存地址、虚拟地址），或逻辑地址，

程序地址等，其对应的存储容量称为虚存容量或程序空间；把实际主存地址称为实（存）地址、物理地址，其对应的存储容量称为主存容量，实存容量或实（主）存空间。当用虚地址访问主存时，机器自动地把它经辅助硬件变换成实地址，并检查该虚地址所指单元内容是否已装入主存。如在主存，就进行访问；如不在主存内，则通过辅助软、硬件把它所在的那块程序由辅存调入主存，然后访问它。上述操作过程和辅助软、硬件对程序员均是“透明”的。所有存储层次均须满足这一点，且辅助软、硬件成本只能占机器总成本中很小比例。

虚拟存储器是存储体系的一部分，存储体系却不一定必须有虚拟存储器。虚拟存储系统与非虚拟存储系统本质的差别在于：前者允许用户用比主存容量大得多的地址空间访问主存，且每次访存都必须进行虚、实地址转换；而对于后者，用户只能以主存容量或以主存中被分配到的那部分空间（比主存容量小）进行访问，且不需进行地址转换。

上述论述从容量需要引出了存储层次，下面则从速度要求来进行分析。为了解决 CPU 与主存速度不匹配而导致 CPU 的性能、效率不能充分发挥的问题，人们提出了多种办法。

一种办法是在 CPU 结构中设置通用寄存器，很多运算可直接在 CPU 的通用寄存器中进行，以减少与主存的通信量，减轻二者速度差距所带来的影响。但由于通用寄存器的编号必在指令有关的编码段中反映出来，通用寄存器数目过大将导致指令编码段太长，给控制器的指令译码网络增加负担；同时，也因优化使用它们而增加编译程序的负担，降低效率。所以，CPU 中的通用寄存器数目不能过大。

另一种办法是采用存储器多体（模块）交叉并行存取以提高主存的等效速度。这种方法对速度的改善是有限的，较深的、复杂的交叉存取所增加的设备量并不能使其性能相应成比例地提高，反而会使性能价格比显著下降。以上两种方法可以同时采用。

第三种办法是用存储层次的方法，即用高速缓冲存储器方式，这是通用寄存器构想进一步发展的必然结果。在 CPU 和主存中间构成“高速缓存-主存”层次（Cache）。要求 Cache 在速度上能与 CPU 匹配，在容量上能存储在一段时间内 CPU 所要用到的程序和数据。同时，还需要有 Cache 地址和主存地址的自动变换、地址映像和调度。在原理上它与虚拟存储器是一样的，不同的只是速度要求高，因而不是软、硬结合而是用全硬件实现。因此，“Cache-主存”层次对应用程序员和系统程序员都是“透明”的。“Cache-主存”层次如图 2-19 所示。

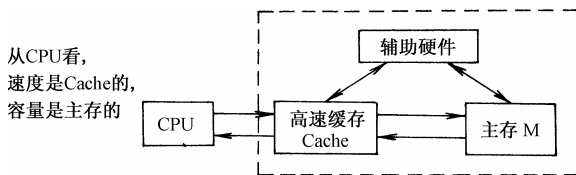


图 2-19 “Cache-主存”层次

“Cache-主存”层次的设计要求是：从 CPU 来看，层次的速度接近于 Cache，但容量是主存的，而每位平均价格应当接近于主存。

由上述“主存-辅存（虚拟存储）”和“Cache-主存”二级存储层次可推广至更多级的存储层次，把配置于机器的各部分的各种存储器有机地联系在一起，构成一个完整的存储体系，如图 2-20 所示。存储体系的系统结构和操作系统的存储管理密切相关，软、硬件的分界面明显地相互交错。系统结构设计者就是要联系具体实现方法，对软、硬件功能分配进行分析和确定。对存储体系的研究一直是系统结构设计者的一个重要任务。

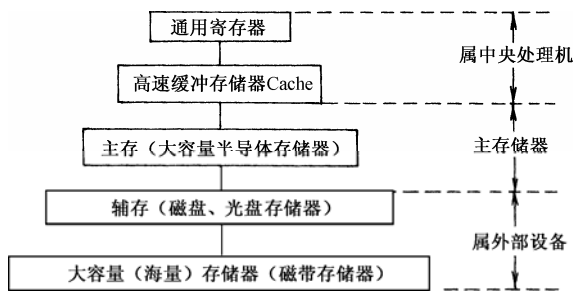


图 2-20 多级存储层次

2.3.3 存储系统的性能参数

多级存储层次是由不同存储技术、不同性能和不同价格的 M_1, M_2, \dots, M_n 级存储器构成。每一级 M_i 在某种意义上附属于较高级 M_{i-1} ，CPU 与 M_1 通信， M_1 再和 M_2 通信，依次类推，如图 2-21 所示。

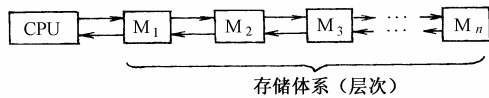


图 2-21 多级存储层次一般表示法

设 c_i 为 M_i 的每位价格， T_{A_i} 为 M_i 的访问时间， S_{M_i} 为 M_i 的存储容量，则 M_{i+1} 与 M_i 间有：

每位价格	$c_i > c_{i+1}$
访问时间	$T_{A_i} < T_{A_{i+1}}$
存储容量	$S_{M_i} < S_{M_{i+1}}$

因此，这个存储层次从 CPU 来看，它在平均速度上接近于 M_1 ，在存储容量上是 M_n ，而每位平均价格接近于 M_n 。在实际设计和实现方面，容量目标容易达到，速度目标较难达到，而价格目标更难达到。

在程序执行时，CPU 发出一个连续的逻辑地址流，在任何时间，这些地址总是以某种方式分布于存储层次内，它总能变换到某个 M_i 的物理地址，如果 $i \neq 1$ ，则地址必须逐级变换到 CPU 能直接访问的 M_1 。逻辑地址的这种定位（变换）需要在 M_i 与 M_1 之间进行相对较慢的信息传送。为提高效率，当 CPU 要访问某个地址的内容时总希望已在 M_1 中准备好。一旦出现被访问信息不在 M_1 中时，程序就暂停执行或被挂起，直到所需信息被调至 M_1 为止。这就要求以后被访问的地址在某种程度上是可预知（预判）的，预判的准确性取决于所使用的算法以及地址映像、变换的方式。这也是存储层次设计成功的主要标志。

评价存储体系主要有三个参数：每位价格 c ，命中率 H ，访问时间 T_A 。现以二级体系（ M_1, M_2 ）为基础进行分析。

（1）每位价格 c 。存储层次的每位平均价格

$$c = (c_1 S_{M_1} + c_2 S_{M_2}) / (S_{M_1} + S_{M_2})$$

式中， c_i 为 M_i 的每位价格， S_{M_i} 为 M_i 以位（bit）计算的存储容量。

为了使 c 接近于 c_2 ，则应使 $S_{M_1} \ll S_{M_2}$ 。但实际上，如果相邻两级容量相差很大，则级

间地址映像与变换很困难,而且会降低层次的效率。

(2) 命中率 H 。命中率是衡量存储体系的重要参数。对于二级存储体系, H 定义为 CPU 发出的逻辑地址能在 M_1 访问到(命中)的概率。 H 可用实验方法求得,以一个典型程序执行或模拟,若逻辑地址流在 M_1 中访问的次数为 R_1 ,因当时在 M_2 而不能在 M_1 访问到的次数为 R_2 ,则命中率

$$H = \frac{R_1}{R_1 + R_2} = \frac{R_1}{R}$$

H 与地址预判算法(即判定下一步访问地址在何处)以及 M_1 的容量有很大关系,预判算法好, M_1 容量大,则 H 也高。 H 当然是越高越好。

不命中率(或称失效率、脱靶) $1-H$,指 CPU 在 M_1 中访问不到的概率。

(3) 访问时间 T_A 。设 T_{A1} 和 T_{A2} 分别是 M_1 和 M_2 的访问时间,则 CPU 访问存储层次的平均时间 T_A 为

$$T_A = HT_{A1} + (1-H) T_{A2}$$

在二级存储层次中,若要访问的单元不在 M_1 中,就必须由 M_2 把包括该单元在内的一个信息块调入 M_1 ,传送信息块所需时间 T_B 称为块交换或块传送时间,因此 $T_{A2} = T_B + T_{A1}$,代入前式,则得

$$\begin{aligned} T_A &= HT_{A1} + (1-H) T_{A2} \\ &= HT_{A1} + (1-H)(T_B + T_{A1}) \\ &= HT_{A1} + (1-H)T_B + (1-H)T_{A1} \\ &= T_{A1} + (1-H)T_B \end{aligned}$$

对“主存-辅存”层次结构,信息块传送需经过 I/O 操作,时间相对慢得多,因此, $T_B \gg T_{A1}$, 即 $T_{A2} \gg T_{A1}$, $T_{A2} \approx T_B$ 。

设 r 为相邻两级的访问时间比,即 $r = T_{A2} / T_{A1}$; 设 e 为存储层次的访问效率, $e = T_{A1} / T_A$ 。 T_A 越接近于 T_{A1} 越好,即访问效率 e 越接近 1 越好。由 T_A 前式可得

$$\begin{aligned} e &= T_{A1} / T_A = T_{A1} / [HT_{A1} + (1-H)T_{A2}] = 1 / [H + (1-H)T_{A2} / T_{A1}] \\ &= 1 / [H + (1-H)r] = 1 / [r + (1-r)H] \end{aligned}$$

由该式可得 $e = f(r, H)$ 函数曲线,如图 2-22 所示。 r 取决于各级存储器的器件和设备特性、 H 与 M_1 容量及替换算法等许多因素。由图 2-22 曲线可得出如下结论:欲使访问效率 $e \approx 1$,在 r 值越大时(即相邻两级访问时间相差越大),则必须要求命中率 H 越高。例如, $r=100$ 时为使 $e > 0.9$,则 $H > 0.998$;而当 $r=2$ 时,则只需 $H > 0.889$ 即可。故在 M_1 和 M_2 速度差很大时,若要有足够高的访问效率,非要有很高的 H 值不可。例如“主存-辅存”层次,由于主存存取速度为几十 ns (10^{-8} s) 至几百 ns (10^{-7} s),辅存存取速度为 ms (10^{-3} s),其速度差达 $10^4 \sim 10^5$ (即 r 值达 $10^4 \sim 10^5$),这就要求命中率 H 很高,访问效率 e 才能高。

但 H 值提高相当不易,虽然增大主存容量,减少主、辅存容量差可提高 H ,但这又会使每位平均价格上升。若在“主存-辅存”之间增加一级(如电子磁盘),使 r 值不过大,则可降低对 H 的要求而获得同样的 e 值大小。

(4) 存储体系的透明性与其性能的关系。一般来说,系统结构透明部分多,程序员使用

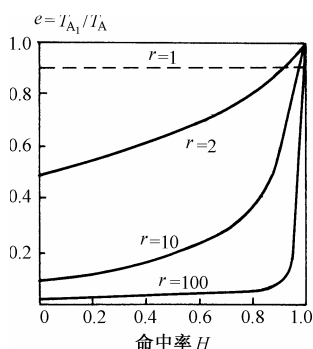


图 2-22 $e=f(r,H)$ 函数曲线

就方便。然而，对于存储体系的干预，不利于根据不同任务调节存储体系的某些参数，不利于任务的切换。因此，不能过分追求存储体系的透明性，只要基本透明即可，某些关键参数应能由程序员控制，这样有利于存储体系优化使用。

2.3.4 程序访问的局部性

存储系统层次结构 (M_1, M_2, \dots, M_n) 中的信息满足三个重要特性：包含性 (Inclusion)，一致性 (Coherence) 和局部性 (Locality)。如图 2-23 所示， M_1 是 Cache，它直接和 CPU 内的寄存器通信，最低层 M_n 包含所存储的信息字。实际上， M_n 可视作虚地址空间，它是全部可寻址的字的集合。

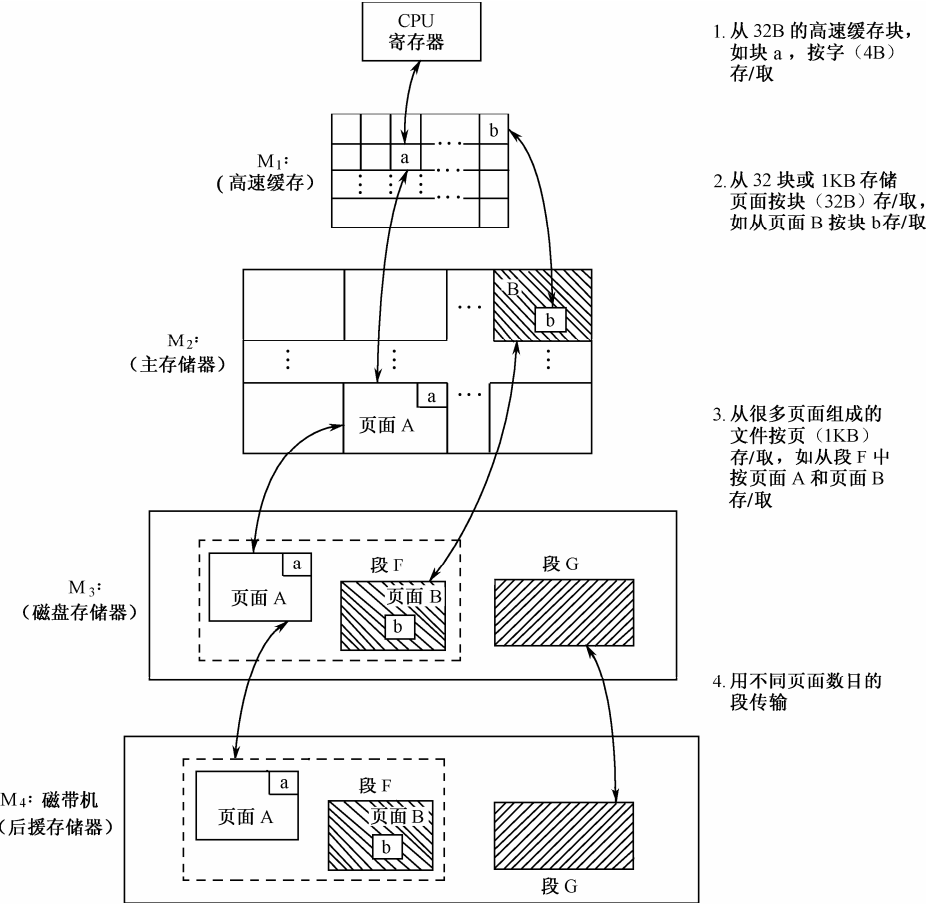


图 2-23 包含性和相邻层之间的数据传输

1. 包含性

包含性可用 $M_1 \subset M_2 \subset M_3 \subset \dots \subset M_n$ 来描述。建立包含关系说明所有信息最初是存放在最低层 M_n 内的。在处理过程中， M_n 的子集复制到 M_{n-1} 层，同样 M_{n-1} 层子集制到在 M_{n-2} 层，依次类推。若在 M_i 层找到某个信息字，那么同一个字的副本在所有低层 $M_{i+1}, M_{i+2}, \dots, M_n$ 中一定可以找到。但是，存放在 M_{i+1} 层中的字在 M_i 层可能找不到。在 M_i 中一个字丢失，因为最低层是后援存储器，所有字都存放在那里，所以在这里总能找到它的副本。

图 2-23 所示例子表明，CPU 和 Cache (M_1) 之间信息的传输按字（一个字是 4B 或 8B，

取决于机器字长)进行, Cache 被分成块 (Cache Block), 也称 Cachet 行 (Line), 每块 32B (8 个字), 块 (如图 2-23 中 a 和 b) 是 Cache 和主存之间数据传送的单位。

主存 (M_2) 分成若干页面, 每个页面有 4KB, 包含 128 块 ($4KB/32B=128$)。页面是磁盘 (M_3) 与主存数据传送的单位。

页面在磁盘内是按段组织的。例如 F 段包含页面 A、页面 B 等, 段的大小可根据用户需要变化。磁盘和磁带 (M_4) 之间数据传送是按文件级处理的, 例如 F 段和 G 段组成一个文件送入磁盘。

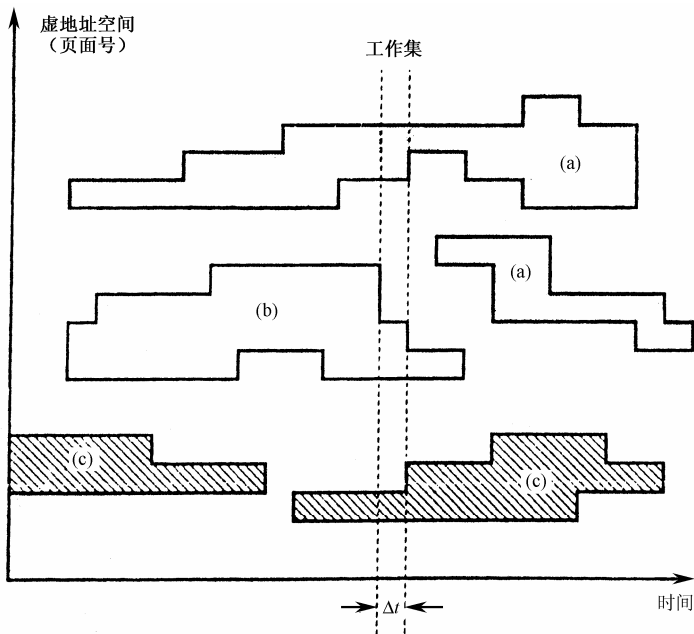
2. 一致性

一致性要求同一个信息项与低层存储器上的副本是一致的。如果在 Cache 内一个字被修改了, 那么, 在主存、辅存上该字的副本也必须立即或最后加以修改, 层次结构必须这样维护。维护层次结构的一致性有两种策略: 一是写直达 (Write-through, WT), 即若在 M_i 中修改一个字, 则在 M_{i+1} 中立即修改; 二是写回 (Write-Back, WB), 即对 M_{i+1} 的修改推迟到 M_i 中已修改的字在替换时或从 M_i 消除时才进行。“Cache-主存”层次替换策略后述。

3. 访问的局部性

存储系统的层次结构是由访问的局部性提出来的。CPU 访问存储器时, 所进行的存取操作在空间、时间和次序上往往集中在一定范围内。Hennessy 和 Patterson 于 1990 年提出一条 90%~10% 规则: 典型程序在其 10% 的机器代码上可能耗费其执行时间的 90%。如嵌套循环时最内层循环就是这种代码。

局部性包括三方面特性: 时间的 (Temporal)、空间的 (Spatial) 和顺序的 (Sequential)。在软件进程的生存期内, 许多存储器页面是动态使用的, 对这些页面的访问也是经常变化的, 但它们遵循一定的访问模式, 如图 2-24 所示。这些访问模式与局部性有关。



(a), (b), (c) 是执行三个软件进程产生的区域

图 2-24 在典型程序跟踪试验中存储器的访问模式

（1）时间局部性。最近访问的指令和数据很可能在不久的将来再次被访问，这是由程序结构所引起的，如迭代循环、进程堆栈、暂时变量或程序等。一旦进入循环或调用子程序，一个代码段将被反复访问。因此，时间局部性往往引起对最近使用区域的集中访问。

（2）空间局部性。它表示一种趋势，指一个进程访问的各项其地址彼此很近，如表操作或数组操作就会对地址空间中某一区域集中访问。某些程序段，如子程序和宏的访问也是如此。

（3）顺序局部性。在典型程序中，除转移类指令外，大部分指令是顺序执行的。顺序执行和非顺序执行的比例大致是 5:1。此外，对大型数组访问也是顺序的。

时间局部性导致近期最少使用（Least Recently Used, LRU）替换算法流行，它还有助于确定相邻层次存储器的容量。空间局部性有助于确定相邻层次间数据传送单位的大小。顺序局部性影响实现最佳调度时耗度（粒度组合）的确定，并对预取技术有重要影响。局部性原理是设计 Cache、主存、虚拟存储器的理论基础。

2.4 输入/输出系统

2.4.1 输入系统

如果把计算机化的信息处理系统比喻为人类的大脑，那么输入系统就相当于人的各种感官，输入系统在计算机系统中扮演着极为关键的角色。输入（Input）一词通常是指预备好输入计算机系统进行处理的信息，也常指把数据送入计算机系统的过程，因此为了使计算机系统发挥它的强大功能，就要有准确、快速的输入设备。为此人们已经开发了各种各样的输入设备，如图 2-25 所示，我们把它分成五大类。

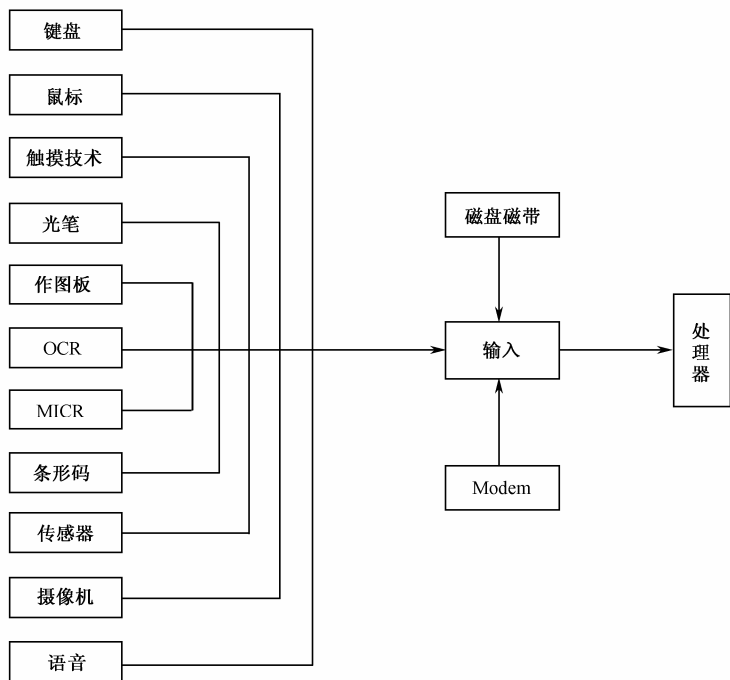


图 2-25 输入设备

(1) 键盘输入类 (Key-driven Devices)。通过操作员击键来输入信息, 这是目前使用最多的手段。在计算机发展早期, 使用穿孔卡与穿孔纸带来输入信息, 而穿孔机也是键盘式的。现在计算机常用一种录入设备, 由录入员把用户信息先录入磁盘或磁带, 再通过磁盘或磁带把信息输入到计算机系统进行处理。这些录入设备称为键入磁盘 (Key to Disk) 或键入磁带 (Key to Tape) 机。它们都是键盘式终端设备。

(2) 指点输入类 (Pointing Devices)。它包括鼠标、光笔、作图板、触摸技术等。当移动鼠标时, 屏幕上的光标就跟着移动, 它们配合先进的窗口 (Windows) 软件, 可以做到光标到哪里, 就处理哪里的程序, 使选择菜单变得十分容易, 在作图中也带来特别的方便。

(3) 扫描输入类 (Scanner Devices)。包括条形码、OCR、MICR 扫描输入等。当用户去图书馆借/还书时, 管理人员用扫描装置扫描一下借书证条形码, 就把用户的证件号输入了计算机, 再扫描贴在图书上的条形码, 就能登记用户借出的书或者注销用户归还的书, 这比手写或键盘输入要快得多。

(4) 传感输入类 (Sensor Devices)。例如, 在遥感卫星或航天飞行器上, 装有摄像机及其他传感器, 可以对地球及其他行星进行探测, 如旅行者 1 号 (Voyager 1) 对木星的探测, 它传回的微弱信号经地面站 (超无线电望远镜的作用) 接收放大后, 送至计算机系统进行处理, 就能得到许多珍贵的资料。

(5) 语音输入类 (Voice Input Devices)。人类把说话的语音直接输入计算机系统。现在, 可以把声音通过话筒变为模拟信号, 再把模拟信号通过调制变为数字编码, 这里最困难的是计算机如何识别、理解这些语音, 这就是语音识别 (Speech Recognition) 技术研究的内容。目前的研究水平大多处于少量词汇、特定人语音的识别阶段, 要让计算机理解人们所说的每一件事, 恐怕还要经历相当长的时间。我国对汉语语音识别的研究也正在进行中。

2.4.2 输出系统

输出系统是计算机实现价值的生动体现, 它能使系统与外部世界沟通, 是计算机与人直接联系的主要渠道, 它把计算机处理的数据转换成用户需要的形式并传送给用户, 或者传送给某种存储设备保存起来, 以便今后再用。这一系统能直接帮助用户大幅度提高工作效率。

不同的应用场合需要不同的输出设备, 人们已经开发了各种各样的输出设备, 如图 2-26 所示, 我们可以把输出设备分成五大类。

(1) 显示器 (Display)。它是利用视频显示技术 (Video Display Technology) 制成的常用输出设备, 由于它显示的信息具有瞬时性, 所以又称软拷贝 (Soft Copy), 目前大量使用的是阴极射线管 (Cathode Ray Tube, CRT), 它的体积和功耗都比较大。为了实现轻、薄、短、小, 现在又出现了平板式液晶显示器 (Liquid Crystal Display, LCD)。

阴极射线管 (CRT) 是目前应用最广泛的显示器件之一, 它最早用于电视接收机, 然后应用于计算机系统, 作为字符、图形和图像显示器。CRT 是电真空器件, 由电子枪、偏转线圈和荧光屏构成。电子枪由灯丝、阴极、栅极、加速阳极和聚焦极组成。CRT 加电后, 灯丝发热, 热量辐射到阴极, 阴极受热发射电子, 电子束经阳极加速和聚焦后射到荧光屏上, 使电子束动能转变成光能, 在屏幕上形成光点 (也称像素点), 由光点组成图像。其电子束应满足以下要求:

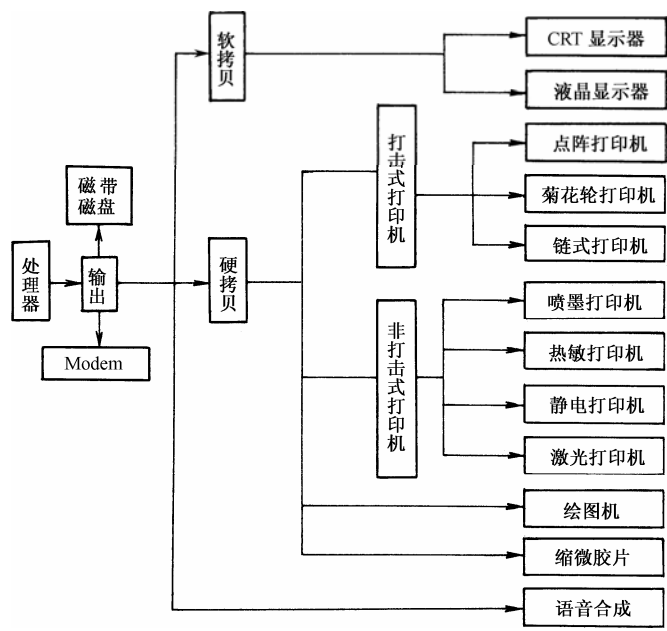


图 2-26 输出设备

① 电子束应有足够的强度和速度，且强度可控，由栅极实现，而栅极由亮度信号控制，根据信号的强弱，控制光点的亮暗，即电子束的强弱。

② 电子束要足够细，使屏幕的分辨率足够高，保证图像清晰。分辨率以“每行像素点数×行数”表示，如 1024×768 表示每行 1024 个像素点（即光点），共有 768 行。分辨率越高，图像越清晰。

③ 电子束运动方向可变，可扫描到荧光屏的任意位置。偏转线圈产生 X 向和 Y 向两个磁场，调动电子束扫描到任意位置。

彩色 CRT 基本原理与单色 CRT 一样，只是彩色 CRT 有红、绿、蓝三基色电子枪（外形上合成一个电子枪），发射三基色电子束，在屏幕上有三基色荧光粉，按三基色叠加原理形成彩色图像。彩色种类的多少，由每个像素点的三基色的位数决定，如每种基色有 n 位二进制数控制其强弱，则可调制出 2^{3n} 种色彩。

电子束打在屏幕荧光粉上的光点只能维持几十毫秒，为了使人眼能看到稳定的图像，就必须用电子束不断地扫描整个屏幕，这个过程称为刷新。每秒刷新次数称为刷新频率或扫描频率。结合人的视觉生理，刷新频率应大于 30 次/秒，图像才不会闪烁。为使图像中运动的事物动作连贯，图像扫描应大于 24 帧（幅）/秒。电视制式是隔行扫描，每秒 50 场，两场一帧，符合上述要求。计算机显示器是逐行扫描，每秒 50 次，50 帧，也符合上述要求，且图像更加稳定。

平板显示器一般是指显示器深度小于显示屏幕对角线 1/4 长度的显示器件，有液晶显示（LCD）、等离子体显示（PDP）、场发射显示（FED）、电致发光显示（ELD）、发光二极管显示（LED）等。

液晶是液态晶体的简称，它是一种有机化合物，在一定范围内，既具有液体流动性，又具有分子排列有序的晶体特征。液晶分子是棒状结构，具有明显的光学各向异性，它本身不

发光,但能调制外照光实现信息显示,因此需要背光源。液晶显示具有工作电压低、功耗小、体轻薄、适合 LSI 驱动,易于实现大尺寸画面显示,显示色彩优良等特点。目前广泛应用的是薄膜晶体管液晶显示器(TFT-LCD)。

彩色等离子体显示是利用惰性气体在一定电压作用下产生气体放电现象而实现的一种发光型平板显示技术。彩色 PDP 技术与荧光灯显示原理相同,利用气体放电产生紫外线,激发光致荧光粉,发射可见光,使用三基色荧光粉实现红、绿、蓝三色,每基色有 256 级灰度(即每基色控制位数为 8 位, $2^8=256$),三基色混色可达到 $2^{24}=256^3$ 种(即 1670 万种)颜色。其结构好比把数十万至数百万个气体微型荧光灯(即放电单元)按一定排列方式设置在两块平板玻璃之间,与 LCD 类似,每个放电单元都有一组电极,并按一定排列形式涂上红、绿、蓝荧光粉。由外部电路按一定方式控制所有放电单元,完成三基色的空间混色,即可实现彩色显示。PDP 的特点是:易于实现大面积显示;全色显示,色纯度与 CRT 相当;视角达 160° ;寿命长。但在功耗、发光效率、对比度、像素间距等方面有待进一步改进,价格目前仍偏高。

(2) 打印机(Printer)。它是以纸为介质,用机、光、电技术制成的打印输出设备。由于它保存的信息具有长久性,所以又称它为硬拷贝(Hard Copy)。打印机以打印字符为主,一次打印一个字符的称为字符打印机(Character Printer);一次打印一行字符的称为行式打印机(Line Printer),简称行打;一次打印一页字符的称为页式打印机(Page Matrix Printer),简称页打。

按工作机构又可将打印机分为击打式(Impact Printer)或非击打式(Nonimpact Printer)。击打式打印机又分许多种,如点阵打印机(Dot Matrix Printer),使用很广,又称针打;菊花轮打印机(Daisy Wheel Printer);链式打印机(Chain Printer)等。非击打式打印机也有若干种,如喷墨打印机(Inkjet Printer)、热敏打印机(Thermal Printer)、静电打印机(Electrostatic Printer)、激光打印机(Laser Printer)等。激光打印机是目前速度最快、质量最高、价格最贵的打印机,通常配置在某个中心,以便大家共享。

(3) 绘图机(Plotter)。绘图机能使用不同性质的纸张,绘出五彩缤纷的高质量图形,它也是一种硬拷贝设备。通常按作图机构把绘图机分为笔式绘图机(Pen Plotter)和非笔式绘图机两类。在非笔式绘图机中,主要有静电绘图机(Electrostatic Plotter)、热敏绘图机(Thermal Plotter)、电子照相绘图机(Photo-electronic Plotter)等。

此外,按照绘图纸笔运动机构的不同,又分为平板式(Flat-bed)、滚动式(Pinch Rolling 或 Friction Roller)、转筒式(Drum)三种。

(4) 影像输出系统(Camera Output Systems)。它是将计算机输出用摄影方法记录下来的系统。一般分两大类:缩微类和录像类,缩微类又称 COM 系统,它包括计算机输出缩微胶卷(Computer Output to Microfilm)和计算机输出缩微胶片(Computer Output to Microfiche)两种,录像类则是将计算机的图像输出用普通录像机记录到录像带(Video Tape)上。

(5) 语音输出系统(Voice Output Systems)。它是用语音对人类理解与感情有意义的声音输出,包括语音和音乐音响输出两大类,它不是用计算机控制的简单录音与放音,而主要是指用合成(Synthesis)来产生声音。

2.4.3 中断系统

在现代计算机系统中,中断系统已成为一个软、硬结合,分布于计算机各部件的一个独立的、重要的系统。它不仅参与管理各种外围设备,而且在人机联系、故障处理、实时处理、

多任务多用户分时操作系统、程序跟踪、调试和监测、用户程序和操作系统的联系、多机系统中各个处理机之间联系以及任务分配等均有不可替代的作用。

中断的定义是：CPU 中止正在执行的程序，转去处理随机提出的请求，待处理完毕，返回原程序继续执行。其核心概念是处理随机提出的请求。

1. 中断源

引起中断的各种事件称为中断源。中断系统的复杂性实际上是由中断源的多样性引发的。中断源可以来自机器内部及处理机本身。中断可以来自硬件，也可以来自软件。

常见中断源有：外设引起的中断、处理机中断（如运算溢出、除数为零、校验错、非法数据格式等）、存储器中断（如非法地址、DRAM 刷新、主存页面失效、数据或地址校验错、访问主存超时等）、控制器中断（如非法指令、未定义操作码、用户程序执行特权指令、堆栈溢出、分时系统中时间片到、操作系统中用户态与特权态切换等）、总线中断（如 I/O 总线出错、MEM 总线出错等）、实时过程控制中断（如采样中断、控制信号中断等）、实时钟定时中断、多机系统中其他处理机发来的中断和控制台开关中断、程序调试断点中断、硬件故障中断、电源故障中断。

为了在响应中断时尽快找到中断服务程序入口，通常根据中断事件的紧迫程度、中断源工作速度、中断源性质等，对众多中断源进行分类，并用硬件实现按类找到中断入口，然后用软件找到该类内某个中断源的入口。通常将中断分为 6 类：重新启动中断、机器校验出错中断、程序性错误引起中断、访问管理程序中断、外部事件中断、输入/输出中断。后 4 类中断源各有一个中断码（或中断向量）用以区分各类中各个具体中断源。

按屏蔽功能，中断源又分为可屏蔽中断和非屏蔽中断。前者在 CPU 关闭中断时，可不响应；后者不管 CPU 是否开放中断，必须响应。

2. 中断优先级

由于中断请求是随机产生的，很可能同时有多个中断请求，因此中断系统必须解决中断优先次序问题。中断优先级的确定是一个涉及计算机系统全局的问题，主要由下列因素决定：

（1）中断源的紧迫性。对于影响全局的一些中断请求必须处于最高优先级，如电源故障、硬件故障、总线请求、校验出错、溢出等，如不及时处理它们会使整个系统无法正常运行。而一些程序性中断，如过程调用、外设中断可处于低级别。

（2）设备的工作速度。快速设备数据存在时间短，必须及时响应以避免数据丢失，优先级可高一些，而低速设备的优先级可安排得低一些。

（3）数据恢复的难易程度。数据丢失后无法恢复的设备其优先级要高，而能自动恢复的设备其优先级可低些。

（4）要求处理机提供的服务量。中断处理及服务时间少的设备其优先级可高一些，而服务量大的设备其优先级可以低一些。

现代计算机系统的中断优先级处理往往由硬件实现。一种是中断优先链式结构，如图 2-27 所示，各中断源的优先级依链上物理位置而定，越接近 CPU（或 IOP）则优先级越高；另一种是利用中断控制器，由中断控制器决定各中断源的优先级，如图 2-28 所示，中断控制器对各中断源采用设置初始化控制命令来选择优先级。

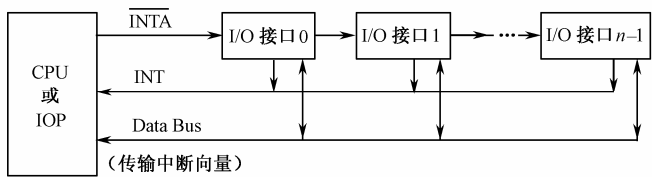


图 2-27 中断优先链式结构

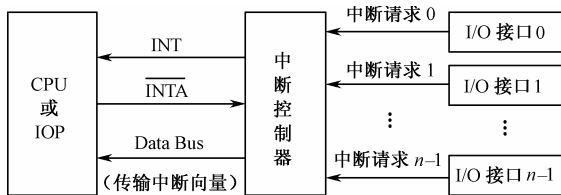


图 2-28 中断优先级并行判优结构

3. 中断系统的软、硬件功能分配

从中断源发出中断请求，到这个请求被处理完成，其过程相当复杂。其中，有的功能用硬件完成，有的功能由软件完成，因此设计一个中断系统时，如何适当分配中断系统的软、硬件功能是关键问题之一。中断系统软、硬件功能分配主要考虑下列两个因素：

(1) 中断响应时间。从中断源发出中断请求到 CPU 响应并转向该中断服务程序入口地址所用的时间称为中断响应时间。这是一个非常重要的指标，尤其在实时控制的计算机系统中，它是关键性指标。

(2) 灵活性。用硬件实现中断系统的某种功能，速度快，但不易修改，且灵活性差；而用软件实现正好相反，灵活性好，但速度慢。

上述两个因素实际上是相互矛盾的。计算机中断系统分布于 CPU、中断控制器、I/O 接口。有关中断响应处理、中断屏蔽、各类中断请求优先级处理、中断向量读取并处理等往往由 CPU 硬件实现；而同种类优先级处理、中断屏蔽、中断向量提供等由中断控制器硬件实现；接收外设请求、中断屏蔽、提供向量可由 I/O 接口硬件实现。至于某个中断源的中断服务则由软件实现，充分体现对不同对象提供不同服务的灵活性。由于目前的中断控制器、I/O 接口均采用 VLSI 芯片，其功能强大，为了给用户更多的选择，弥补纯硬件缺少灵活性的缺憾，都将它们设计成了可编程芯片，即用户可通过初始化程序为芯片设置相应的控制字、命令字，从而达到选择工作方式、选择屏蔽字、选择优先级排列、设置中断向量等目的，在硬件快速的特性上体现了某种软件的灵活性，取得“双赢”的效果。必须指明，对于可屏蔽中断请求，中断系统往往设置三道屏蔽门，即 I/O 接口、中断控制器、CPU 三级递进屏蔽功能，而且均可用软件予以开放和禁止。至于不同种类的中断请求的优先级处理往往体现在 CPU 硬件的实时流程中。

2.4.4 通道处理机和 I/O 处理机

大型计算机系统所带外设种类众多、数量庞大，工作方式和速度差别较大，如继续用程序查询、中断、DMA 方式管理外设，会引起下列问题：

(1) 所有外设的 I/O 均由 CPU 承担和管理，则 CPU 用于用户程序计算时间大受影响，不能充分发挥 CPU 的计算潜力。

（2）外设虽然很多，但同时工作机会不是很多，特别是 DMA 方式下，而每个外设均需配置相应的 I/O 接口，I/O 接口数量过多，利用率却不是很高。

为了使 CPU 摆脱繁重的 I/O 负担，实现 I/O 与 CPU 真正并行操作，使用通道处理机实现 I/O 管理是大型计算机系统常用的方法。通道处理机是一台能够执行有限个 I/O 指令并能与多台外设共享的专用处理机。大型计算机系统可以有多个通道，一个通道可以连接多个设备控制器，而一个设备控制器可以带一个或多个外设。这样就形成了 CPU、通道、设备控制器、外设 4 级层次结构的 I/O 系统。

通道 8 个功能如下：

- ① 接受 CPU 发来的 I/O 指令，选择指定外设与通道连接。
- ② 执行通道程序。
- ③ 给出外设有关地址。
- ④ 给出主存缓冲区首址。
- ⑤ 控制外设与主存缓冲区之间传输数据长度，判断数据传送是否结束。
- ⑥ 执行数据传送结束时要进行的操作。
- ⑦ 检查外设工作状态并保存。
- ⑧ 在数据传输过程中进行格式变换。

通道应该能执行一组通用指令，具有实现上述功能的硬件。通道主要硬件是寄存器组（数据缓冲寄存器、主存地址寄存器、传输字节计数器、通道命令字寄存器、通道状态字寄存器）和控制逻辑（分时控制、地址分配、数据传送、数据装配和拆卸）。通道对外设的控制是通过 I/O 接口或设备控制器进行的，通道与控制器之间采用标准 I/O 接口连接。

1. 通道的工作过程

通道完成一次数据传输的过程如图 2-29 所示。主要过程分以下三步进行：

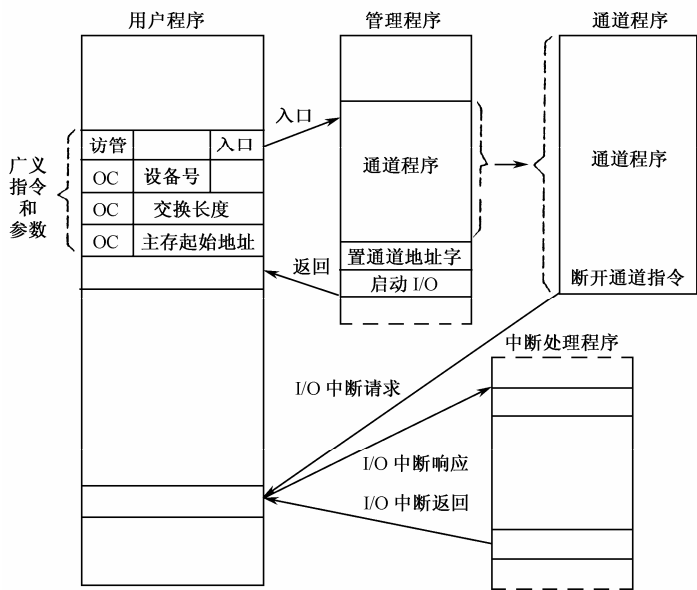


图 2-29 通道完成一次数据传输的主要过程

(1) 在用户程序中使用访管指令进入管理程序，由 CPU 通过管理程序组织一个通道程序并启动通道。在多任务或多用户系统中，I/O 指令属于特权指令，用户程序不允许使用。如要进行 I/O 操作，必须通过一条请求 I/O 的广义指令进入操作系统，调用操作系统的管理程序使用外设。

(2) 通道处理机执行通道程序，完成数据 I/O 工作。通道执行通道程序与 CPU 执行用户程序是并行的，如图 2-30 所示。

(3) 通道程序结束后向 CPU 发中断请求，CPU 响应中断请求，第二次进入操作系统，调用管理程序进行处理，进行必要的登记等工作，如是故障、出错则进行例外情况处理，然后 CPU 返回用户程序继续进行。

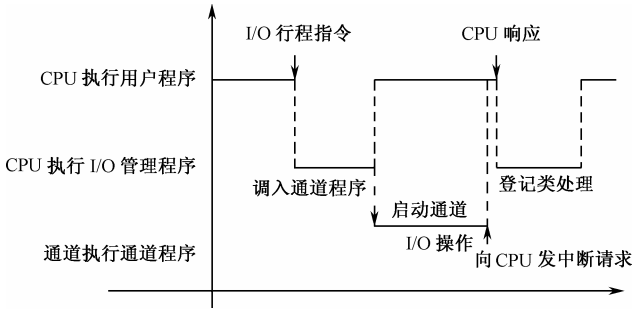


图 2-30 通道执行通道程序与 CPU 执行用户程序的时间关系

2. 通道种类

通道分为三种类型：字节多路通道（Byte Multiplexor Channel）、选择通道（Selector Channel）和数组多路通道（Block Multiplexor Channel），如图 2-31 所示。

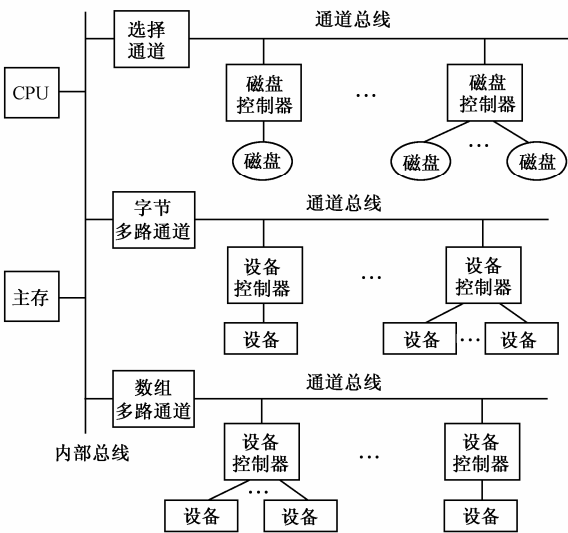


图 2-31 三种类型的通道与 CPU、设备控制器和外围设备的连接关系

(1) 字节多路通道是一种简单的共享通道，主要为多台中、低速外设服务。它采用分时方式使通道上各外设轮流占用一个很短时间片（小于 $100\mu s$ =传输 1 字节）。它包含多个子通

道，每个子通道有一个设备控制器，通道控制部分是公共的，所有子通道共享，而每个子通道都有自己独立的一套寄存器组，每个子通道至少有一个字节缓冲寄存器、一个状态控制寄存器和固定地址译码逻辑。

(2) 选择通道用于高速外设，如磁盘，需要很高的数据传输率。通道在一段时间内单独为一台外设服务，直至该外设数据传输工作全部结束为止。因此，选择通道就其工作形态而言，可视为只有一个以成组方式工作的子通道，只有一套完整硬件，它逐个为物理上连接的几台高速外设服务。硬件主要包含寄存器组（数据缓冲寄存器、外设地址寄存器、主存地址计数器、传输字节数计数器、状态控制寄存器等）、格式变换部件和通道控制逻辑。

(3) 数组多路通道是把上述两种通道特点组合在一起而形成的。每次选择一个外设传送一个数据块（对磁盘和磁带数据块通常为 512B），并轮流为多个外设服务，因此数组多路通道可视为成组工作的高速多路通道。磁盘读/写操作分三步：第一步定位，找磁道；第二步找扇区；第三步读/写数据。前二步为机械动作，需几十毫秒，第三步只需十几微秒。所以，数组多路通道向某台外设发出定位命令后就立即从逻辑上与该设备断开，直到完成定位，找到扇区，才传送数据。所以通道在为一个磁盘传送数据时，另有多个磁盘在定位或找扇区，因此通道数据传输速率和硬件利用率都很高，但控制较复杂。目前，大部分高性能计算机系统都采用数组多路通道。

3. 通道的流量分析

通道流量也称通道吞吐率、通道数据传输率等，指通道在单位时间内传送的最大数据量，单位是字节/秒。通道最大流量与通道工作方式（即通道类型）、通道选择设备所用时间 T_S 、传送 1 字节所用时间 T_D 、通道上连接外设数 P 、每个外设传送字节数（如 P 台外设都同时工作，每台外设都传送 n 字节） n 有关。

(1) 字节多路通道。每台外设传送 n 字节时，所需时间

$$T_{\text{Byte}} = (T_S + T_D)Pn$$

最大流量

$$f_{\text{max} \cdot \text{Byte}} = \frac{Pn}{(T_S + T_D)Pn} = \frac{1}{T_S + T_D}$$

实际流量是通道上所有外设数据传输率之和，即

$$f_{\text{Byte}} = \sum_{i=1}^P f_i$$

(2) 选择通道。连接 P 台外设，每台外设传送 n 字节，所需时间

$$T_{\text{select}} = T_S P + T_D n P = \left(\frac{T_S}{n} + T_D \right) Pn$$

最大流量

$$\square f_{\text{max} \cdot \text{select}} = \frac{Pn}{\left(\frac{T_S}{n} + T_D \right) Pn} = \frac{1}{\frac{T_S}{n} + T_D}$$

实际流量是该通道上所有外设流量最大者

$$f_{\text{select}} = \text{MAX}\{f_i\} \quad (i=1, 2, \dots, P)$$

(3) 数组多路通道。有 P 台外设，每台外设传送 n 字节，由于一次工作传送一个数据块，

数据块是 k 字节, $k < n$ 。所需时间

$$T_{\text{block}} = T_S \cdot \frac{n}{k} \cdot P + T_D P n = \left(\frac{T_S}{k} + T_D \right) P n$$

最大流量

$$f_{\text{max} \cdot \text{block}} = \frac{P n}{\left(\frac{T_S}{k} + T_D \right) P n} = \frac{1}{\frac{T_S}{k} + T_D}$$

实际流量类同于选择通道, 是所有外设中流量最大者

$$\square f_{\text{block}} = \text{MAX}\{f_i\} \quad (i=1, 2, \dots, P)$$

为了保证通道正常工作, 通道实际流量 \leq 最大流量。当两边相等时, 通道处于满负荷工作状态。在实际设计通道最大流量时, 应当留有余地, 即最大流量略大于实际流量, 否则所有外设请求集中出现时, 有可能丢失数据。

【例 2-1】 字节多路通道连接 $D_1 \sim D_5$ 共 5 个外设, 这些外设分别在第 $10\mu\text{s}$, $30\mu\text{s}$, $30\mu\text{s}$, $50\mu\text{s}$, $75\mu\text{s}$ 向通道发一次请求, (该时间包括 T_S 和 T_D) , 问:

(1) 通道实际流量和工作周期是多少?

(2) 如果该通道最大流量等于实际流量, 并设传输速率高的设备优先级也高, 5 台外设 在 0 时刻同时提出请求, 以后时间里按各自速率连续工作, 画出该通道时间关系图。

(3) 从图上发现什么问题? 如何解决?

解: (1) 实际流量

$$\square f_{\text{Byte}} = \frac{1}{10} + \frac{1}{30} + \frac{1}{30} + \frac{1}{50} + \frac{1}{75} = 0.2 \text{ MB/s}$$

工作周期

$$t = 1/f_{\text{Byte}} = 5\mu\text{s/B}$$

(2) 时间关系图如图 2-32 所示。

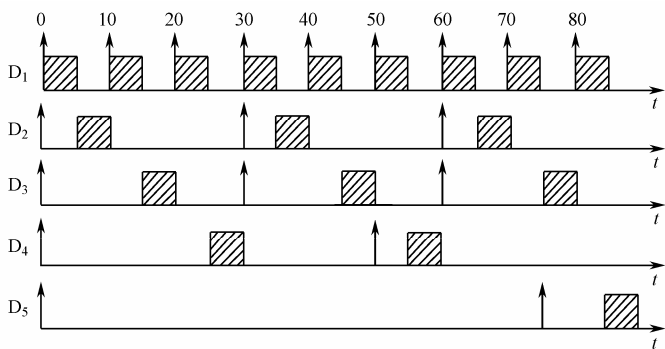


图 2-32 字节多路通道响应设备请求和为设备服务的时间关系图

(3) 从图中可见, D_5 外设第一次请求未被响应, 直至第 $85\mu\text{s}$ 时才响应 D_5 , 所以第一次传送数据丢失。解决方法如下: ① 增加通道最大流量, 保证所有请求能得到响应; ② 动态改变优先级, 如在 $30\mu\text{s} \sim 70\mu\text{s}$ 之间临时提高 D_5 优先级 (即将 D_5 第一次请求与第二次请求时间间隔由原来的 $75\mu\text{s}$ 改成 $60\mu\text{s}$); ③ 给 D_5 增加一个数据缓冲寄存器。上述三种方法取其一即可解决 D_5 第一次丢失数据问题。

4. I/O处理机

通道不能看成独立的处理机，因为通道指令很简单，只有面向外设的控制和数据 I/O 指令，且没有独用的大容量存储器。在数据 I/O 过程中，CPU 仍承担相当多的工作。例如，调用一次 I/O 操作的前处理和后处理由 CPU 完成（见图 2-30）；外设或通道出现异常是通过中断由 CPU 处理；I/O 传送过程中数据格式转换、码制转换、校验等仍由 CPU 处理；I/O 传送中数据、设备管理等操作系统工作由 CPU 实现。基于上述原因，CPU 资源并不能得到充分利用，CPU 执行用户程序和 I/O 操作不能实现完全的并行操作。对于流水线计算机和向量计算机来说，频繁的 I/O 操作使高性能 CPU 无法充分发挥其运算功能，严重影响运算速度。所以，引入 I/O 处理机（器）（Input Output Processor, IOP），由 IOP 全权负责 I/O 和管理外设，实现 CPU 执行用户程序与数据 I/O 的完全并行操作，极大地提升系统运算速度和 CPU 资源利用率。使用 IOP 的计算机典型的结构如图 2-33 所示。

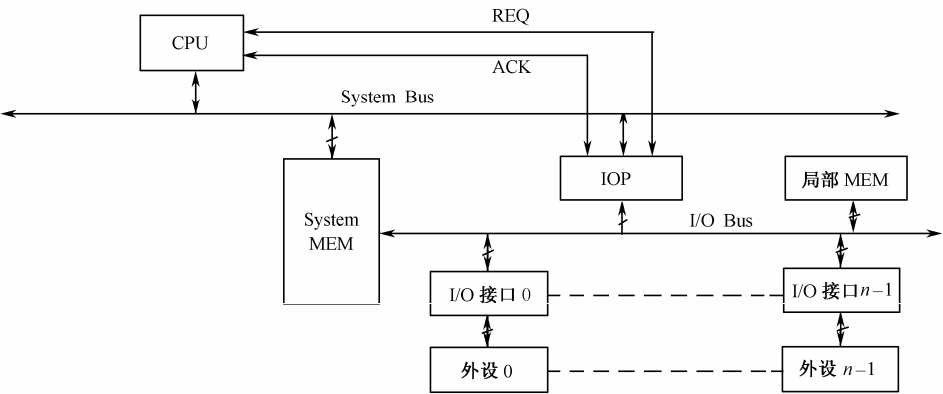


图 2-33 使用 IOP 的计算机典型结构

IOP 有自己的指令系统，除能完成通道全部功能外，还有码制特换、数据校验和纠错、故障处理、文件管理、诊断和显示系统状态、人机对话、网络通信、数据库和知识库管理以及根据需要分配给它的其他任务。IOP 除了 I/O 功能外，还具有运算功能和程序控制功能，如同一个 CPU 一样。另外，IOP 具有自己的专用存储器（局部 MEM），不必通过主存（System MEM）就能完成与外设的数据交换，从而进一步提高整个计算机系统的性能。CPU 与 IOP 是通过 System MEM 内开辟一个缓冲区（信箱）实现信息交互的。CPU 要输出的数据先存于“信箱”，然后放弃 System Bus，向 IOP 发出请求（REQ），IOP 予以响应（ACK），掌握 System Bus，从“信箱”取走数据存于局部 MEM，再撤销 ACK，CPU 恢复掌握 System Bus，撤销 REQ，执行原中止的程序，与此同时，IOP 可通过 I/O Bus，与某个外设实现数据输出。至于输入，其过程类似，不再叙述。系统初始化时，CPU 在主存某特定区域设置 IOP 初始化命令字，选择 IOP 的工作方式等，IOP 到该区域取命令字，设定工作方式。

根据是否共享主存，可把 IOP 分为两大类：

- （1）共享主存的 IOP。每个 IOP 有一个小容量的局部 MEM，IOP 要执行的管理程序在主存中，为各个 IOP 共享，只有某个 IOP 用到才加载到自己的局部 MEM。
- （2）不共享主存的 IOP。每个 IOP 有自己的大容量局部 MEM，存放各自的管理程序，因此减少了对主存的压力。

根据是否共享运算部件和指令控制部件，可把 IOP 分为两大类：

- (1) 合用同一个运算部件和指令控制部件的 IOP。这种 IOP 价格较低，但控制较复杂。
- (2) 有独立运算部件和指令控制部件的 IOP。由于 VLSI 技术高度发展，此类 IOP 已成为主流，且往往具有大容量局部 MEM，更具有独立性。

根据各种计算机系统的不同要求，IOP 组织结构有下列 4 种：

- (1) 有多个 IOP，从功能上加以分工；
- (2) 并行计算机中以 IOP 为主处理机，除承担 I/O 任务，还运行操作系统；
- (3) 使用相同型号处理机，一个为 CPU，另一个为 IOP；
- (4) 计算机系统使用专门设计的廉价的 I/O 微处理器，如 Intel 公司的 8089, 80168 IOP 等。

第 3 章 存储系统结构

存储器主要用于存放计算机的程序和数据。存储系统是指存储器硬件以及管理存储器的软、硬件。对存储系统的基本要求是大容量、高速度、低成本。单一的存储器难以满足计算机系统的要求，所以提出了多层次、硬件和软件相结合的存储体系结构。本章将围绕存储系统的基本组成原理、地址映像与变换、替换算法及其实现等展开，着重介绍并行存储器、高速缓冲存储器（Cache）、虚拟存储器的原理和系统结构。同时，简述存储保护和控制。

3.1 地址映像与变换

3.1.1 程序的定位

CPU 只执行在主存内的程序，在辅存中的程序调入主存时，要进行程序定位，即如何把程序的逻辑地址变换成实际的主存物理地址。这依靠定位辅助硬件以及确定程序在主存中位置的算法来实现的。程序定位的目的是提高主存空间利用率，及时释放主存中的无用区。程序定位可以全部或部分地在程序生成的各个不同阶段或程序执行时进行。

目前，程序定位分为加基址（界）方式和地址映像方式。

加基址方式有以下两种：

（1）静态定位。调入的程序块在实主存空间位置固定，即其实存基址固定，只有待实存中相应位置的原存程序执行完毕，方可调入，如图 3-1 所示。原实存内有 A 道、B 道程序在运行。C 道程序在辅存，其长度 $n \leq m$ (A 道长度)。A 道在实存的基址为 a ，也是 C 道的基址。

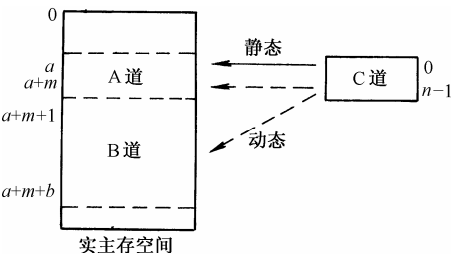


图 3-1 基址定位方式

C 道程序只能在 A 道程序执行结束时才能调入，且其位置只能从基址 a 开始。如 B 道程序执行比 A 道快，且 B 道实存空间大于 C 道空间，在静态定位方式下，C 道程序不能进入 B 道空间，即 C 道基址不能从 a 改成 $a+m+1$ 。

（2）动态定位。调入的程序块在实主存的空间位置浮动，即其实存基址可视实际状况予以调整，如图 3-1 所示。在上例中，B 道先于 A 道执行结束，在动态定位时，C 道基址就可由 a 变为 $a+m+1$ 。基址值修改用特权指令实现。动态定位可提高主存利用率，也相应减少了 C 道程序等待时间。但加基址方式不便于多任务对子程序共享，地址映像方式则易于实现这点。

地址映像方式有以下三种。

（1）段式管理。段式存储是把一个程序分解成多个在逻辑上形成整体、相互独立或基本独立，且定义清楚的模块。这些模块可以是各种功能的子程序或分程序，也可以是各种数据结构的数据集合。这些模块大小可以不同，甚至预先不知道。每个模块为一段，段内地址从 0 开始编址。该段调入主存，只需由操作系统赋予该段一个基址（即该段在主存中的起始地

址)，由基址与段内地址组合形成该段每个单元在主存中的实地址。

主存按段分配的存储管理方式称为段式管理。段式管理需有“段表”指明各段在主存中位置，见图 3-2。段表组成如下：

- ① 段号。各段用户名称或数据结构格式名称或段的编号。
- ② 段起点。该段在主存的起始位置（即基址值）。
- ③ 装入位。该段是否已装入主存。“1”表示装入，“0”表示未装入，装入位随该段是否调入主存而变化。
- ④ 段长。该段长度，可用于判断访问地址是否越界。

段表还可以有其他项目。段表本身也是一个段，常驻留主存，也可存于辅存，需要时再调入。

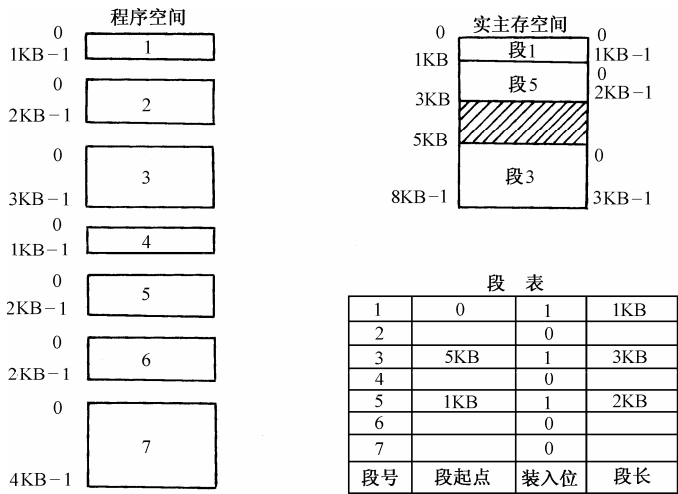


图 3-2 段式管理

分段可使编程模块化、结构化，从而可以并行编程，也便于修改。用户程序以段链接形成的程序空间可与主存的实际容量无关。分段还便于多任务共用已在主存内的程序和数据，公用的程序或数据按段存于主存，不需重复存储，只需在每个任务的段表中用公用段的名称和同样的基址值即可。分段还便于以段为单位实现存储保护。例如，可设计成常数段只读不写；可变数据段只能读、写，而不能作为指令执行；子程序段只能执行，不能修改；有的过程段只能执行，等等。一旦违反规定就产生软件中断，这对发现程序设计错误和非法使用很有效。

(2) 页式管理。页式存储是把主存空间和辅存（或虚存）中的程序空间按固定大小（一般为 512 字节至几千字节）分成若干页。主存按页顺序编号（即物理页号）。每个程序也各自按页顺序编号（即逻辑页号或虚存页号），而各页可以装入主存中不同的页面位置。

主存按页分配的存储管理方式称为页式管理。页式管理也需要“页表”，如图 3-3 所示。页表组成如下：

- ① 虚存页号（逻辑页号）。它是程序空间中各页面的编号。由于虚存页号纯粹是顺序编制，故在页表内该项可有可无。
- ② 主存起点（实存页号）。每页在主存中的起始位置（即基址值），也可用主存页号代

替（因为每页容量固定）。

③ 装入位。该页是否已装入主存。“1”表示装入，“0”表示未装入。装入位随该页是否调入主存而变化。

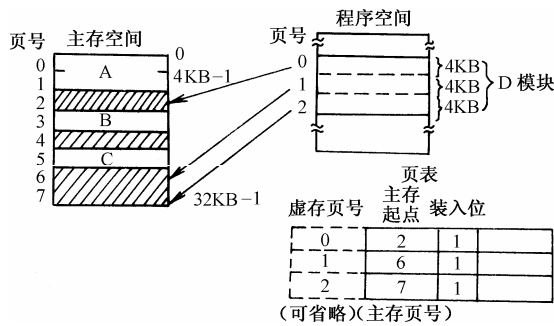


图 3-3 页式管理

页表还可有自定的所需项目。图 3-3 是一个页式存储管理例子，主存中已有 A, B, C 三个模块，现 D 模块有 12KB，每页容量为 4KB，页表反映了 D 模块在主存的情况。所以，页表反映了程序空间到主存空间的映像。页表由操作系统和存储体系配合，根据主存运行情况建立，此过程对程序员完全透明。

(3) 段页式管理。它是分段和分页相结合的一种存储管理方式，具有二者的综合优点，为大、中型计算机广泛采用。它是把实存等分成页，程序按模块分段，每段再按实存页同样大小的容量，分成若干页面。这样，每道程序（任务）是通过一个段表和一组页表进行定位。段表中的每行对应一个段，行内有一个指向该段的页的起始地址，以及该段的控制保护信息，由页表指明该段各页在主存中的位置以及是否已装入，已修改等。由于采用分页，它与纯段式的主要区别在于段的起点不是任意的，必须是实存中页面的起点。

多用户的每个用户（每道程序）需有一个用户标志号（称为基号），用以指明该道程序的段表起点存于哪个基寄存器中。这样，程序地址格式如图 3-4 所示（对于多用户，基号必须有；对于单用户，基号可有可无）。图 3-4 所示的例子将主存分成 32 个页面，已有 A, B, C 三道程序占用主存，图内用阴影表示。现有 D 道程序要调入，它分成三段，0 段分二页（页号为 0, 1）；1 段分二页（页号为 0, 1）；2 段分三页（页号为 0, 1, 2），共有 7 页。主存现空余 8 页，分散分布（主存空页号为 1, 2, 4, 5, 7, 8, 10, 11），按容量完全可以接纳 D 道程序。如采用纯段式，D 道的 2 段需三页，比主存现在的任何空余大而无法调入（主存现在最大空余为二页）。采用段页式则可调入。D 道各段各页在主存的位置如图 3-4 所示。

段页式地址变换举例见图 3-5。现程序地址（虚地址）为 D 道 2 段 1 页内的 d ，变换过程如下：

- ① 由基号经基寄存器找出 D 道段表起始地址 S_D ；
- ② “ $S_D + \text{段号 } 2$ ” 找到 D 道段表内 2 段这一行，取出页表起始地址 c ；
- ③ “ $c + \text{页号 } 1$ ” 找到 D 道 2 段页表内 1 页这一行，取出实存页号 8；
- ④ 将实存页号 8 与程序地址内的页内位移 d 拼接，得到实存地址。

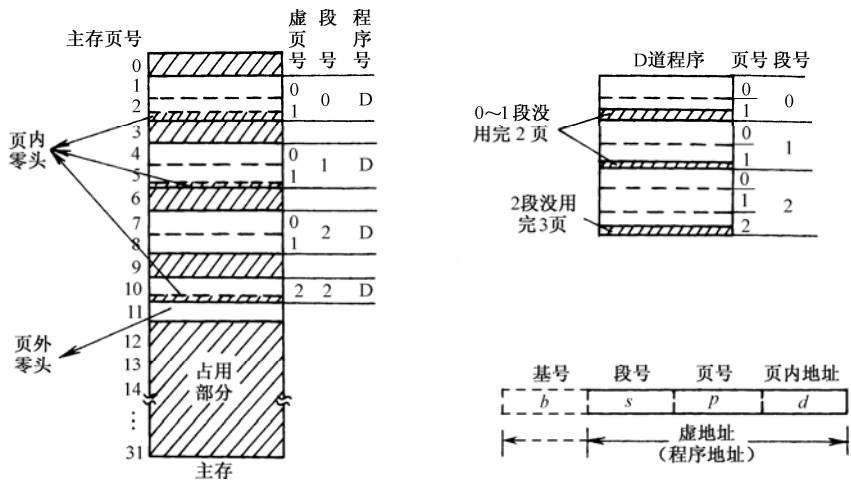


图 3-4 段页式管理

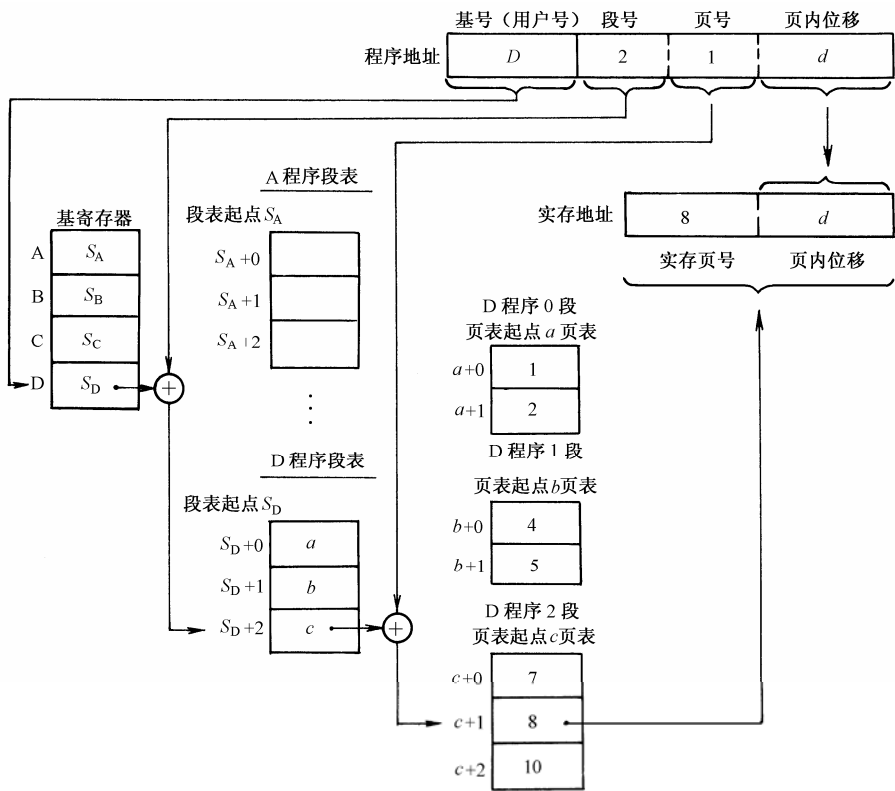


图 3-5 段页式地址变换举例

由上例可看出，段页式地址变换至少需查两次表（段表和页表）。段、页表构成表层次。表层次在纯页式也会有，因为页表的长度大于主存的页面是完全可能的，从而页表分存于主存中不连续的各页内。如按“页式管理”所述，由页表起始地址加页号查表方法就会出错（前述页式查表方法是基于整个页表连续存储）。例如，虚地址为 22 位，页内地址为 9 位，则页表就需要 $2^{N_v} = 2^{13}$ 行，页面大小为 $2^{N_r} = 2^9$ ，若页表每行为一个存储单元。 $2^{N_v} / 2^{N_r} = 2^{13} / 2^9 = 2^4 = 16$ ，

该页表要分存于 16 个页面。采用表层次结构可使每个页表大小限制在一页之内。

用树的概念可得出表的层次数 i 和 N_v, N_r 的关系式为

$$2^{N_v} = (2^{N_r})^i, \quad i = \frac{\log_2 2^{N_v}}{\log_2 2^{N_r}} = \frac{N_v}{N_r}$$

因为 i 必为整数，所以向上取整， $i = \lceil N_v / N_r \rceil$ 。

对于上例来说， $i = \lceil 13/9 \rceil = 2$ ，需二级页表。从这种意义上讲，也可把段页式视作为克服纯页式中页表超过页面而增加的层次。

如果页表中一行需要 B_e 个存储单元，若 B_e 为 2 的幂（即 $2^0, 2^1, 2^2, 2^3 \dots$ ），则 B_e 需用 $N_e = \log_2 B_e$ 地址位表示，则表的层次数

$$i = \lceil N_v / (N_r - N_e) \rceil$$

如上例中，若页表一行为 8 字节， $B_e = 8 = 2^3$ ，则 $N_e = 3$ ，故表的层次数 $i = \lceil 13 / (9 - 3) \rceil = 3$ ，原来为二级，现在则要分成三级。

采用表层次技术不能减少页表（第二级表）空间，反而增加段表（第一级表）空间。它的好处在于段表驻留主存，页表小部分在主存，大部分在辅存，需要时调入，从而减少每道程序所占用的主存空间。

在分页虚拟存储体系中，将虚存（辅存）空间划分成 2^{N_v} 个页面，每个页面容量为 2^{N_r} 个存储单元，但实存（主存）只有与虚页同样大小的 2^{n_v} 个页面。一般情况， $2^{N_r} \ll 2^{N_v}$ ，因此，怎样把一个大的程序空间压缩、映像到小的实存空间，是地址映像和变换所要解决的问题。

地址映像（或称定位算法）是指每个虚页按什么规则（算法）装入（定位于）实存；地址变换是指程序按照映像关系装入实存后，在程序运行时，虚地址如何变换成对应的实地址。

如虚地址 N_s 由 N_v 和 N_r 拼接而成；实存地址 n_p 由 n_v 和 n_r 拼接而成，如图 3-6 所示。由于程序的起始地址必为页面的始点，故虚存页内地址 N_r 与实存页内地址 n_r 是相等的（即 $N_r = n_r$ ），不必映像与变换。因此，地址映像与变换就是 N_v 与 n_v 之间的关系。因为实存远小于虚存，因此，实存中每个页面必须能与多个虚页相对应。不同的映像方式对应的虚页数量是不同的，但至少应是虚存总页面数除以实存总页面数，即 $2^{N_v} / 2^{n_v} = 2^{n_d}$ （ $n_d = N_v - n_v$ ）。当同时有两个以上的虚页要进入同一个实页时，会发生页面争用（实页冲突），这是选择映像方式时要考虑的重要因素。下面分别讨论不同的映像方式。

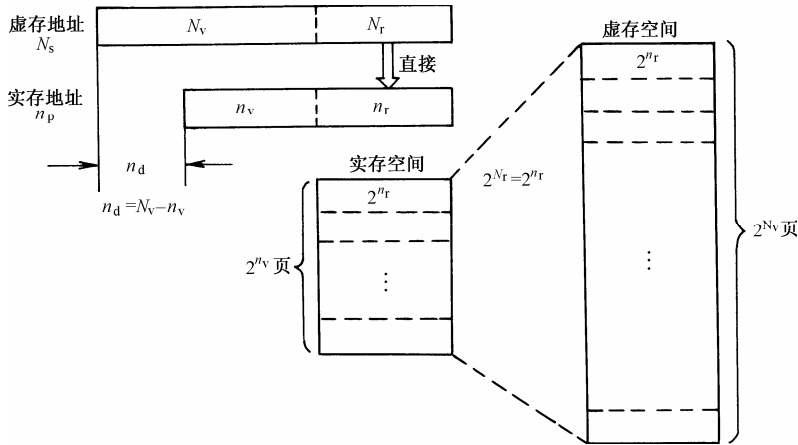


图 3-6 虚存与实存的对应关系

3.1.2 全相联映像及其变换

全相联映像的定义是，任何虚页可映像到实存任何页面位置，如图 3-7 所示。从地址来看，就是任何 N_v 可对应于任何 n_v ，只有当一个程序同时调入的页数超过 2^{n_v} 个页面（即超过实存容量）时，两个虚页才可能争占一个实页，但这种现象很少出现。因此全相联映像的突出优点是实页冲突概率最小。它的地址变换方法有两种。

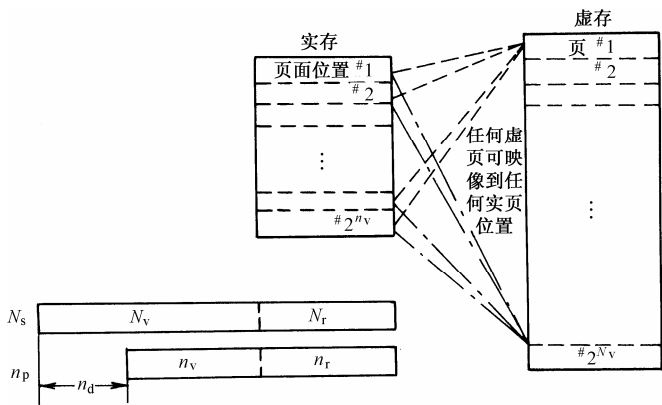


图 3-7 全相联映像

1. 页表法

这是一种直观的地址变换方法。设 $N_v=4$ 位， $n_v=2$ 位，如图 3-8 所示。在实存的 RAM 中有一个页表，共有 2^{N_v} 个单元（本例为 $2^4=16$ 个单元）。若 $N_v=0100$ ，至页表的第 0100 行取信息，判断装入位为“1”，表示该页已在实存内，取出 $n_v=11$ ，将此实页号与 N_r 拼接，得到实存地址 n_p 。若装入位为“0”，表示该虚页未调入实存，就需要由辅存把该页调入。

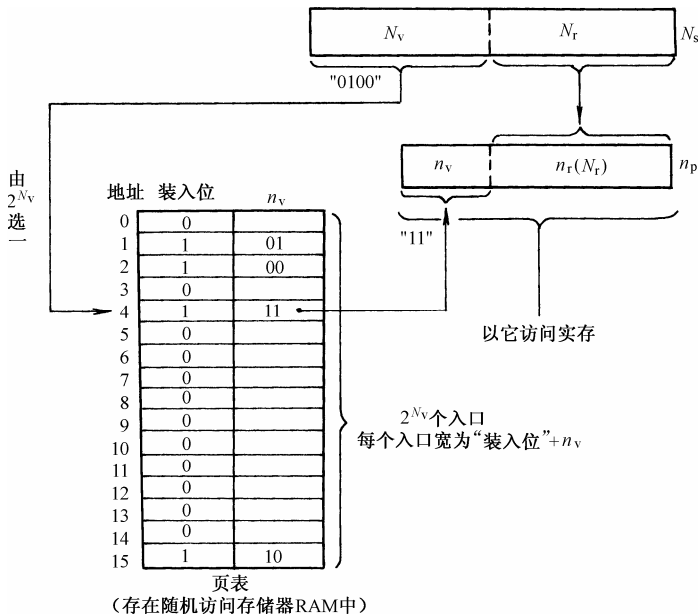


图 3-8 页表法

2. 目录表法

页表中装入位为“1”的最多只有 2^{n_v} 行（本例为 $2^2=4$ 行），这是 2^{n_v} 因为实存内只有 2^{n_v} 个实页。因此页表中装入位为“0”的行的 n_v 空闲。故可把页表压缩成 2^{n_v} 行，每行由 N_v+n_v 位构成，取消装入位，形成目录表，如图 3-9 所示。将目录表存于相联存储器中（相联存储器是按存储内容进行访问，而不是按地址访问），地址变换时，按虚地址中的 N_v 至相联存储器查找（也称相联比较），找到，则表示该虚页已调入实存，从这一行的 n_v 部分取出实页号，与 N_r 拼接成实存地址 n_p ；找不到，则表示该虚页未调入实存。目录表所需容量比页表小（页表容量可能大到需采用表层次技术），但使用相联存储技术造价较高。

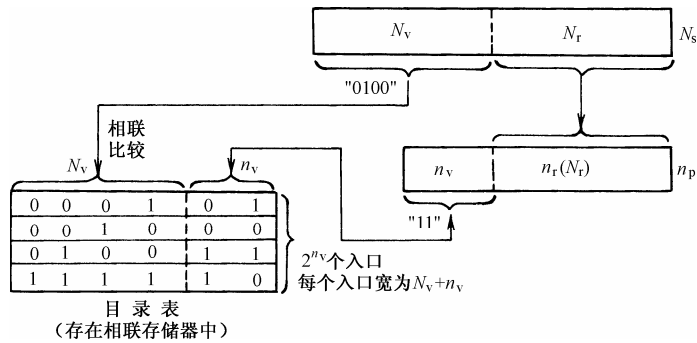


图 3-9 目录表法

3.1.3 直接映像及其变换

直接映像的定义是每个虚页只能映像到实存的一个特定页面，如图 3-10 所示。这相当于把虚存空间按实存空间分块，而块内各页直接映像到实存的相应页。

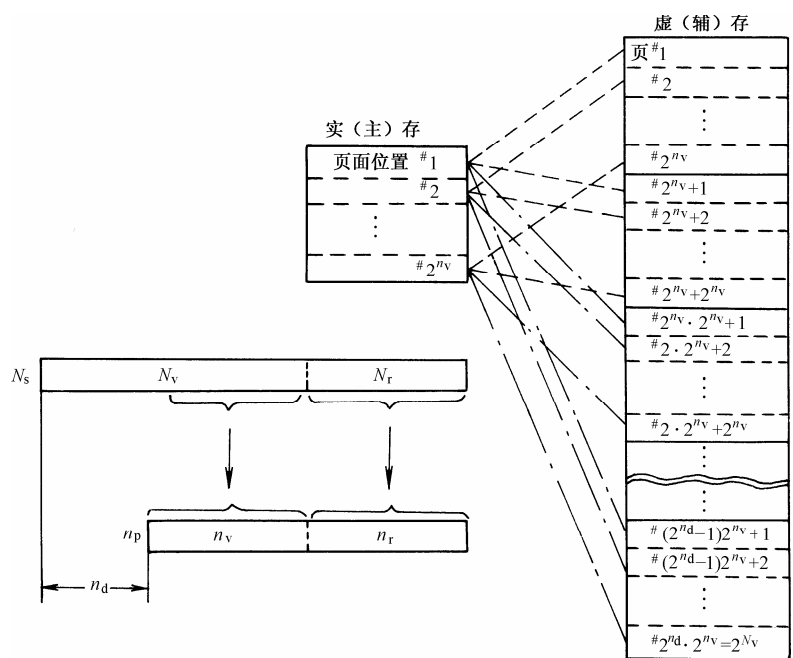


图 3-10 直接映像

直接映像的优点是，在地址变换时，只需将 N_v 中与 n_v 对应的部分与 N_r 拼接即可，如图 3-10 所示。用此拼接地址访问实存时，还需判断该虚页是否已在实存中，因为能映像到该实页位置的虚页有 2^{n_d} 个，所以要用 n_v 去查表，以 2^{n_v} 作为查表地址，将表内对应的 n_d 读出，然后与 N_v 中的 n_d （即该虚地址的 n_d 值）比较，如果相等，表示该虚页已在实存中，否则页面失效。用拼接地址 n_p 访问实存的同时，进行 n_d 的查表，如判断出页面失效，则按 n_p 访问实存作废，这样节省了时间，地址变换过程见图 3-11。直接映像的缺点是实页冲突概率高，实存利用率低。

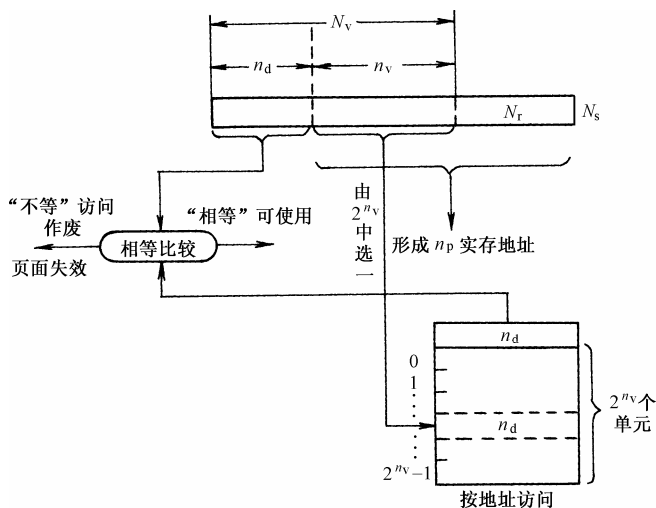


图 3-11 直接映像地址变换

3.1.4 组相联映像及其变换

组相联映像的定义是，各组之间是直接映像，组内各页间是全相联映像。它是全相联和直接两种映像方式的结合。如图 3-12 所示，整个实存为一块，分成 Q 组， $Q=2^q$ （图示为两组， $q=1$ ， $Q=2^1=2$ ），每组分成分 S 页， $S=2^s$ （图示为 4 页， $s=2$ ， $S=2^2=4$ ），实存共有 2^{n_v} 个页（图示为 $2^3=8$ 页）。实存地址 n_p 中的 n_v 由组号 q 、组内页号 s 组成。虚存分成与实存同样大小的 2^{n_d} 块（图示 $n_d=1$ ，虚存分 $2^1=2$ 块），每块与实存对应分成同样大小的组和页。虚地址 N_s 由块号（ n_d ）、组号（ q' ）、页号（ s' ）组成的 N_v 再拼接页内地址 N_r 组成。其中， q' 、 s' 字段的宽度和位置与实存地址的 q 、 s 是一致的。从图 3-12 可见，虚存第 0 组只能进入实存第 0 组，虚存第 1 组只能进入实存第 1 组（直接映像），而虚存的 0, 1, 2, 3 及 8, 9, 10, 11 页可进入实存 0, 1, 2, 3 中任意一页（全相联映像），但不能进入实存的 4, 5, 6, 7 页。例如虚存 0 页已在实存 0 页，现虚存 8 页要调入，若是“直接映像”，就要发生实页冲突，而对组相联映像，则虚存 8 页可进入实存 1, 2, 3 中任意一个页面，所以组相联映像的实页冲突概率比“直接”低得多。 S 值越大，实页冲突概率就越低。

在实存容量和页面大小已定的前提下（即 $n_v=q+s=\text{常数}$ ），对 $S=2^s$ 和 $Q=2^q$ 值进行选择，可取得不同效果。 S 值越大（即 s 增大， q 下降），实页冲突概率越低，地址变换越复杂，当 S 值大到等于实存页面数 $S=2^{n_v}$ （即 $s=n_v$ ， $q=0$ ）就变成了全相联映像。 S 值越小（即 s 下降， q 增大），实页冲突概率越高，地址变换越易实现，当 S 值小到等于实存 1 页（即 $s=0$ ， $q=n_v$ ）

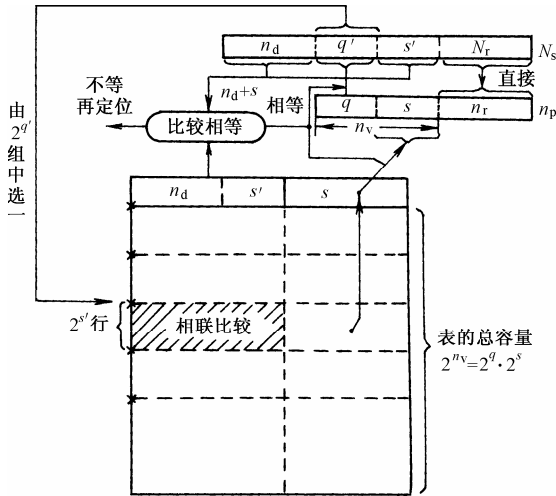


图 3-13 组相联映像地址变换

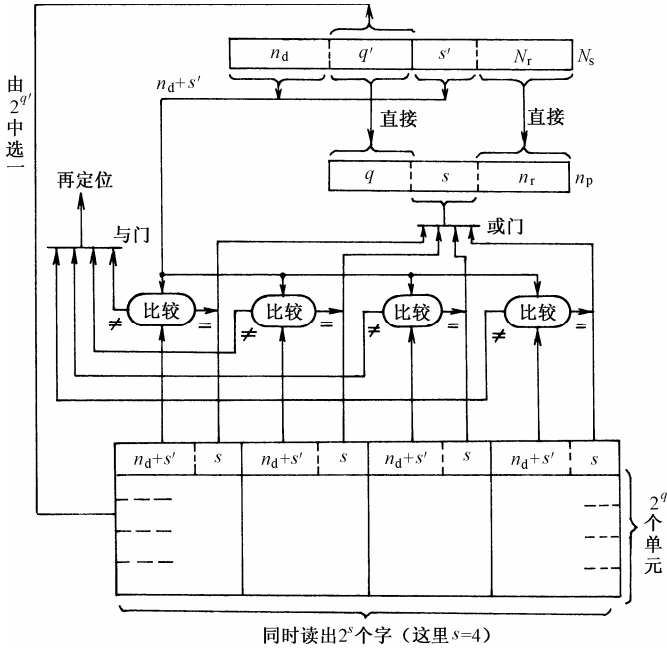


图 3-14 单体多字并行存储器地址变换

为了降低相联比较的复杂度，一般实存的段容量（即段的数量）比较小，而每个段内的页容量（即页的数量）很大，这也是与一般组相联映像不同之处。段相联地址变换过程如图 3-15 (b) 所示。虚存地址和实存地址都由三部分组成：段号、段内页号和页内地址。由于虚存的容量比实存容量大得多，因此虚存地址中的段号 s' 要比实存中段号 s 长得多，而段内页号和页内地址一样长。由于虚存的段内页与实存的段内页是直接映像方式，因此虚存地址的段内页号和页内地址可以直接送实存地址，而实存地址中的段号 s 需要从段表中查出来。段表由虚存段号 s' 和实存段号 s 组成。每个虚存段号 s' 下有 sZ 个实存段号 s 。当访问实存时，首先用虚存地址中段号 s' 与段表中虚存段号字段进行相联比较，如有相等的，再用虚存

地址中的段内页号 Z 按地址访问相应段表中的实存段号部分，取出实存段号 s 。最后把该 s 与虚存地址中段内页号 Z' 和页内地址 W' 拼接得到实存地址。如相联比较没有相等的，则发出失效（不命中）请求。另外，在段表中与实存段号相联系的还有一个有效位，用来反映虚存段号与实存段号之间映像关系是否有效，即已在实存内的这个页是否是有效副本。当虚存段号 s' 在段表内进行相联比较不命中时，则发生失效，若进行替换，则要把被替换的段内所有页的有效位都清除，以便新的段调入。段相联映像和变换的主要优点是段表比较简单，实现成本较低。缺点是当发生失效时，由于段内页容量比较大，因此被作废的页数也比较多。

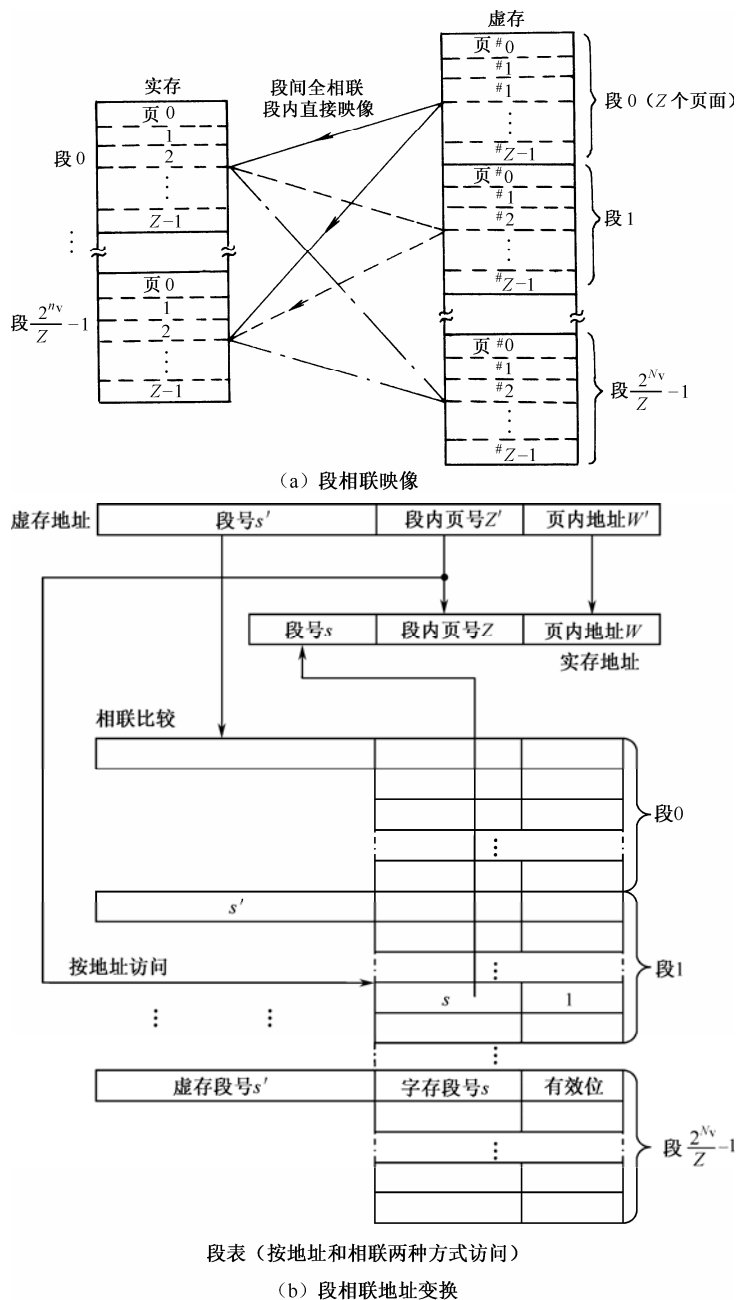
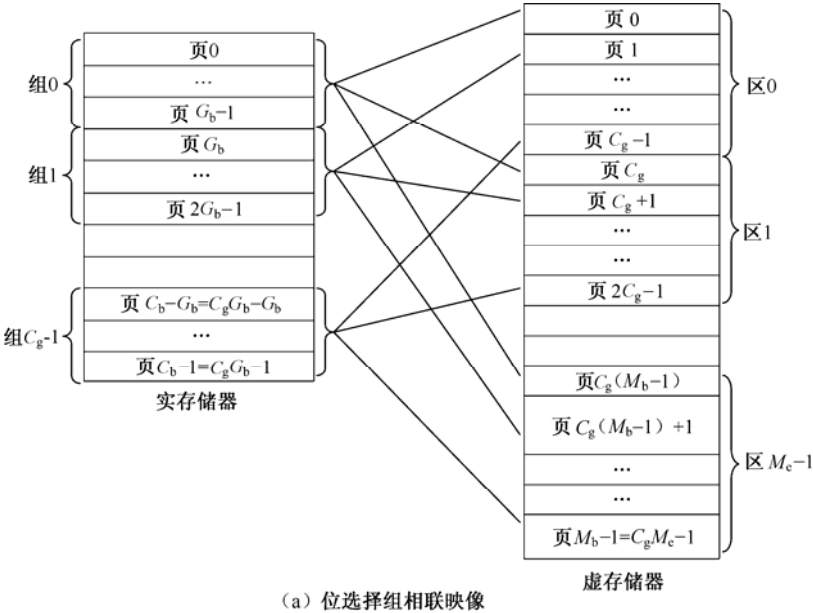


图 3-15 段相联映像与地址变换

3.1.6 位选择组相联映像及其变换

位选择组相联映像又称节相联，其映像方式如图 3-16 (a) 所示。实存分组，而虚存不再分组，虚存除了分页之外，还按照实存的组容量 C_g 分区。虚存中每个区内页的数量与实存的组的数量相等。虚存中的页与实存中的组是直接映像，而在实存组内是全相联映像。在组相联映像中，虚存中一个组与实存中的一个组之间是多个页到多个页的映像，而在位选择组相联映像中，虚存中一个页到实存的一个组之间是一个页到多个页的映像。因此映像关系明确，较易实现。另外，从数据分布情况分析，对于组相联映像中，虚存中的几个连续页映像到实存中可能也是连续的，而对于位相联映像，虚存中的连续页映像到实存肯定是不连续的，它们被分散到实存的各个组内，只要实存中一个字不跨越两个页，在实存内部的页与页之间分布是否连续对实存的正常工作无多大影响。



(a) 位选择组相联映像

图 3-16 位选择组相联映像及地址变换

位选择组相联地址变换方式的一种实现方法如图 3-16 (b) 所示。由于虚存每个区中的页容量与实存中的组容量相等，而且它们之间采用直接映像，因此虚存地址中的区内页号可以直接作为实存地址中的组号。同理，也可以直接得到页内地址。地址变换就是通过查页表得到实存地址的组内页号。另外，页表内有效位是标记区号 E 与页号 b 建立的映像关系是否成立，有效位的使用和管理可参照直接映像。位选择组相联地址变换与一般组相联地址变换过程基本相同。

上述介绍的多种地址映像方式及其地址变换实现方式虽说都是在虚（辅）存和实（主）存层次内，但对于主存和 Cache 层次也完全适用，其不同仅在于名称不同，虚存地址为主存地址，主存地址为 Cache 地址，页号为块号，仅此而已。

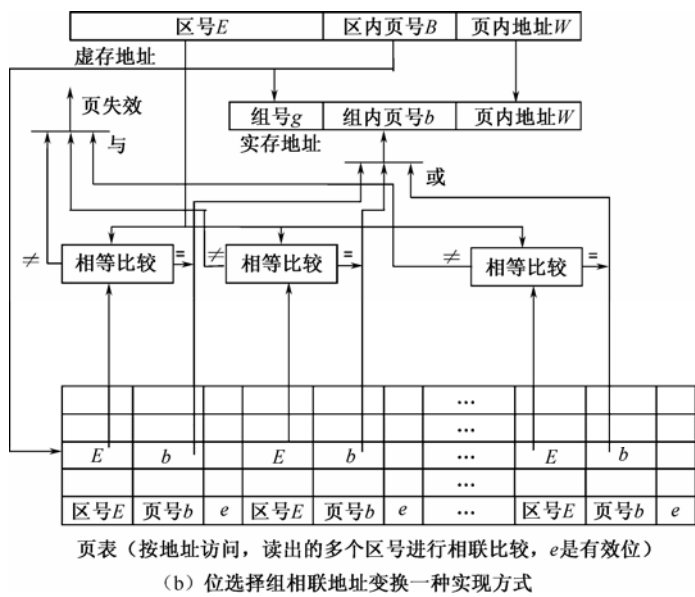


图 3-16 位选择组相联映像及地址变换（续）

3.1.7 对标志表的分析

从上述各种映像和地址变换中可以看出，地址变换需要用到各种“映像表”，这些表统称为标志表，表 3-1 列出了三种常用标志表的特性。对标志表的分析不能单从表的总容量来判断其优劣。因为相同容量的相联存储器比 RAM 成本高而速度较低，因此要根据硬件的速度、容量、成本以及对存储体系的要求、实页冲突概率等综合考虑。但是，目录表法比页表法造价低。

表 3-1 标志表特性比较

项 目		全相联查表法	全相联目录表法	组 相 联
行数（入 口 数）		2^{N_v}	2^{n_v}	2^{n_v}
入口状况	直接访问	2^{N_v}	0	2^q
	相联访问	0	2^{n_v}	2^s
相联比较的位数		0	N_v	n_d+s'
每个入口的窄宽度（注）		n_v+1	N_v+n_v	n_d+2^s
最小存储容量		$(n_v+1)2^{N_v}$	$(N_v+n_v)2^{n_v}$	$(n_d+2^s)2^{n_v}$
可以采用的存储器硬件		按地址访问的单体单字存储器	按内容访问的相联存储器	按地址访问加上按内容访问的存储器。当 s 值小时，用按地址访问的单体多字存储器

注：实际上还需要加某些控制信息位。

地址映像的本质是“压缩”空间，即大的程序空间“压缩”到小的实存空间，而页表法到目录表法也是一种压缩。因此用压缩概念便于从理论上探索效率更高的映像方式。

3.1.8 散列概念在地址变换中的应用

全相联目录表要求能按内容 N_v 查找，使用相联存储器芯片价格高、容量小、速度较慢。

用按地址访问的 RAM 芯片组成目录表的存储器，价格低、容量大、速度快。在对这种存储器中的信息按内容查找时，可以有顺序查找、对分查找和散列查找等多种方法，其中散列（Hashing）方法速度最快。

散列法的基本思想是，让内容（也称关键字 Key）与存放该内容的地址 A 构成散列函数关系，即 $A=H(\text{Key})$ 。存入时，将 Key 和其他内容存入地址为 $A=H(\text{Key})$ 的存储单元中。查找时，按给定的 Key 经散列函数变换成 A，然后依 A 为地址去访存，从中取出的 Key' 与原 Key 比较：相符者，则该单元内其余内容为所要查找部分；不相符，则表示出现散列冲突。具体过程见图 3-17。

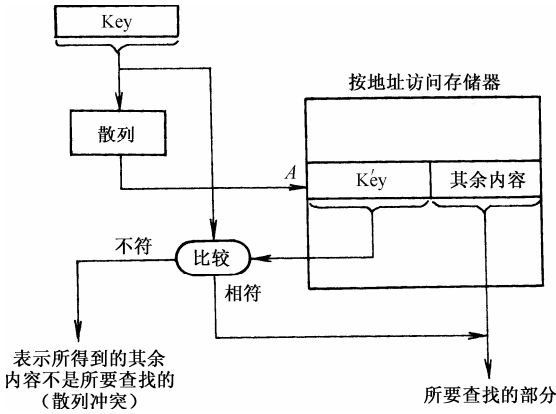


图 3-17 散列法地址变换过程

散列函数有如下几种：

（1）相除法。将 Key 除以一个常数后，或者取其商数，或者取其余数作为 A。一般采用余数法，因为，如果除数为 C，余数必在 $0\sim C-1$ 范围内（即地址 A 在 $0\sim C-1$ 范围内）。如果 C 为质数，所得余数可较均匀地分布于 $0\sim C-1$ 之间。因为同余数的数肯定不是唯一的，所以会出现多个关键字对应同一个地址的现象。例如 $C=23$ ， $\text{Key}_1=73$ 的地址 $A_1=H(73)=4$ ， $\text{Key}_2=119$ 的地址 $A_2=H(119)=4$ ，二者一样，即 $A=H(\text{Key}_1)=H(\text{Key}_2)$ 。这种多个关键字对应同一个地址的现象称为散列冲突。对散列法而言它是不可避免的，即关键字和地址不是唯一对应的。

（2）相乘法。将关键字乘以某个常数后，取乘积中某一段作为散列地址 A，这也会有散列冲突，如图 3-18 所示。

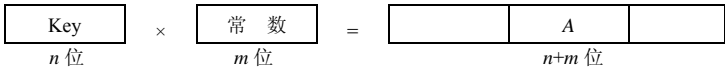


图 3-18 相乘法

（3）折叠法。将关键字按 g（应是质数）分成几段，不足 g 的高位补零，然后对这些段进行相加或按位加，得到 g 位宽的散列地址 A，如图 3-19 所示，关键字为 12 位，以 $g=5$ 分段，然后进行按位加，得到 5 位的散列地址。这种方法也有散列冲突。

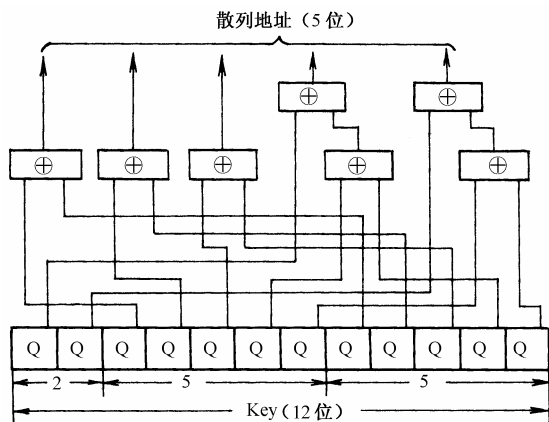


图 3-19 折叠法地址变换

对散列冲突有两种处理方法：① 把具有 Key_1 关键字的内容放在 A 地址单元，而把具有 Key_2 关键字的内容放在 $A+1$ 单元内，这在硬件上便于实现，也便于查找；② 将具有 Key_2 关键字的单元放在 $A_2=H[H(Key_2)]$ 地址单元内，即再进行一次散列。

散列概念用于存储体系的地址变换时，为满足速度要求，用硬件实现。图 3-19 即为按位加折叠散列硬件原理。至于相除法、相乘法都可以经 ROM 或 PAL（GAL）实现，随着 VLSI 发展，散列硬化很容易实现。

3.2 替换算法及其实现

在存储体系中，当出现页面失效时，或者主存（实存）所有页面已经全部被占用而又出现页面失效时，按照哪种算法（规则）来替换主存中某页，这就是替换算法问题。

替换算法的确定要有利于提高存储体系的性能，主要是提高命中率。显然，二次页面失效之间的时间间隔越大，则命中率越高。其次是替换算法是否易于实现。

在存储体系中，很多地方用到替换算法。在“主-辅”层次中用于主存页面替换；在“Cache-主”层次中用于 Cache 页面替换；在地址变换的“快表-慢表”层次中用于快表内容的替换。它们的原理是相同的，实现方法有所不同，可能是全软的，也可能是全硬的或者软硬结合的。

3.2.1 替换算法的分析

替换算法主要有下列 4 种：

（1）随机算法（RANDom, RAND）。用硬件或软件随机产生被替换的页号。由于没有利用实存使用的“历史”信息，不能反映程序的局部性，故命中率很低。这种算法基本不使用。

（2）先进先出算法（First-In First-Out, FIFO）。先进入实存的页先被替换。该算法虽然利用了实存使用的“历史”信息，让最早进入实存的页被替换出去，但没有正确反映程序的局部性。因为最早进来的页，也许是目前或最近一段时间内要经常用到的页，所以该算法不能反映实际使用需求。

（3）近期最少使用算法（Least Recently Used, LRU）。把近期最久未访问的页替换出去。这种“近期”是指过去了的近期，该算法是根据过去的近期使用状况预测未来近期中哪一页

可能不被使用，而替换出去，故能比较准确地反映程序的局部性，命中率有所提高。但毕竟是指过去了的近期，不能真正预测未来，所以有一定的局限性。

(4) 优化替换算法 (Optimal Replacement Algorithm, OPT)。预测各页今后使用时刻，选择其中时间间隔最长的页替换出去。例如，主存有 A, B, C, D 4 个页面，在时间 t_i 时刻要将页面 E 调入，如果在 t_i 之后，A 在 t_j 时刻用到，B 在 t_k 时刻用到，C 在 t_l 时刻用到，D 在 t_m 时刻用到，它们与 t_i 的时间间隔分别为 $t_j - t_i$, $t_k - t_i$, $t_l - t_i$, $t_m - t_i$ ，选出其值为最大者的页面用 E 替换（如 $t_l - t_i$ 值最大，则用 E 替换 C）。这种算法是最合理的。但怎样在 t_i 时预测到 t_j , t_k , t_l , t_m 时各页面使用状况呢？只有让程序在机器中运行两次，第一次是分析地址流，第二次才是真正运行。这是不现实的，因为既费时又费事，在编译和执行过程中几乎不可能取得为实现 OPT 所需的信息。因此，OPT 是一种理想的算法，是衡量各种算法优劣的标准。

有的替换算法增加判别被替换的页是否修改过（被写过）的功能。如果没有修改过，在替换时不必先写回辅存（因为辅存中有其副本）。如果修改过，则必须先将它写回辅存，然后调入新页。因此，被替换的页的选择还可加上“未访问过”、“已访问过但未修改过”、“已访问过且已修改过”等条件。

为了评价各种替换算法的优劣，可用模拟方法，用典型的页地址流，分析各种算法的命中率，由其高低来评价算法的好坏。当然，影响命中率的因素除了替换算法之外，还有地址流状况、页面大小、地址映像方式和实存容量等。

设有一个程序，共有 5 页，其页地址流如图 3-20 上部所示。分配给该程序的实存有三页，现分别用 FIFO, LRU, OPT 替换算法推算这三页的使用情况，其中按所用算法选中为替换页的用星号 (*) 标记，具体过程见图 3-20。从图 3-20 可见，FIFO 命中 3 次，命中率 $H=3/12=0.25$ ；LRU 命中 5 次，命中率 $H=5/12=0.417$ ；OPT 命中 6 次，命中率 $H=6/12=0.50$ 。这个过程与实际运行是相符的，FIFO 命中率最低，LRU 命中率接近 OPT。

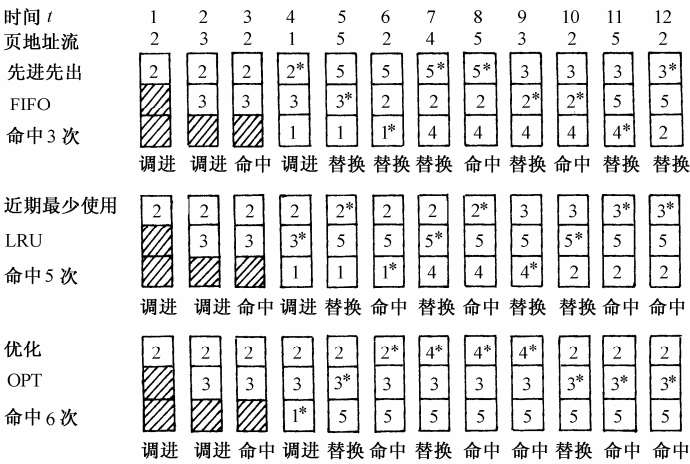


图 3-20 替换算法过程

对某种页地址流，LRU 与 FIFO 的命中率一样差。例如，循环程序所需页数大于分配给它的实存页数，就会出现这种状况。如图 3-21 所示，若循环程序需 4 页，而实存只有三页，则 LRU 与 FIFO 命中率均为零，OPT 命中三次。在本例中，LRU 和 FIFO 均连读、频繁地出

现页面失效，几乎没有有效运算时间，这种现象称为颠簸。颠簸使系统效率显著下降，它是评价存储管理和存储体系性能的一个重要标志。从图 3-20 可看出，如果分配给此程序的实存页数增加一页，则命中率可显著增大。

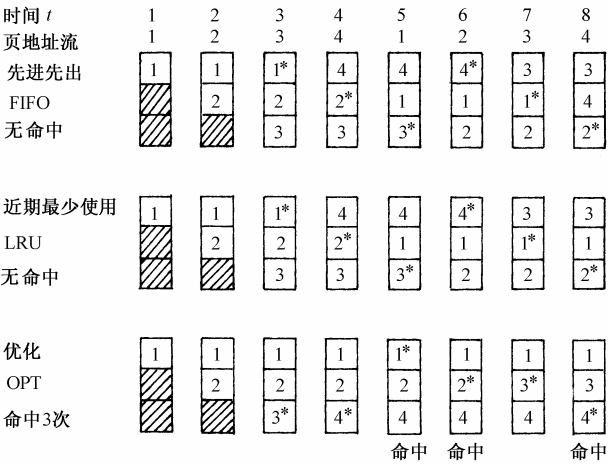


图 3-21 循环程序所需页数大于实存页数的替换算法过程

由于影响命中率的因素很多，要对所有因素进行模拟工作量非常大。为此提出了若干能优化存储体系设计、减少模拟工作量的分析模型，其中堆栈处理技术可用于分页系统，适用于堆栈型替换算法。

堆栈型替换算法的定义如下：设 A 是长度为 L 的任何页地址流， t 为已处理过 $t-1$ 个页面的时间点， n 为分配给该地址流的实存页面数， $B_t(n)$ 表示在 t 时间点 n 页实存的页面集， L_t 表示到 t 时已遇到过的地址流中相异页的页数，如果替换算法具有下列包含性质：

$$\begin{aligned} n < L_t \text{ 时, } B_t(n) &\subset B_t(n+1) \\ n \geq L_t \text{ 时, } B_t(n) &= B_t(n+1) \end{aligned}$$

则此算法为堆栈型的。

由于 LRU 算法在实存中保留的是 n 个最近使用过的页面，它们又总是包含在 $n+1$ 个最近使用过的页面里，所以 LRU 是堆栈型算法。同样，OPT 也是堆栈型算法，而 FIFO 不是堆栈型算法。

对于堆栈型算法。只需采用堆栈处理技术对页地址流模拟处理一次，就能得到此页地址流在不同实存容量时的命中率，从而大大节省对分页系统的分析时间。图 3-22 是用堆栈处理技术对 LRU 的一个典型页地址流的分析。堆栈处理技术的要点是：将实存的页以堆栈方法压入：凡有命中的页，则弹至栈顶；凡无命中页，则将调入的页压入栈顶，其余依次下压，将最底部的页替换出去（压出）。从图 3-22 可看出，分配给该页地址流的实存页面数 n 不同，则命中率 H 也不同。各个 n 值的命中率 H 如表 3-2 所示。

表 3-2 不同实存页面数的命中率

实存页面数 n	1	2	3	4	5	>5
命中率 H	0.00	0.17	0.42	0.50	0.58	0.58

从表 3-2 可看出，LRU 的命中率随分配给该程序的实页数增加而单调上升，其他堆栈型算法也是如此。而 FIFO 是非堆栈型算法，实页数增加有时反而可能降低命中率。

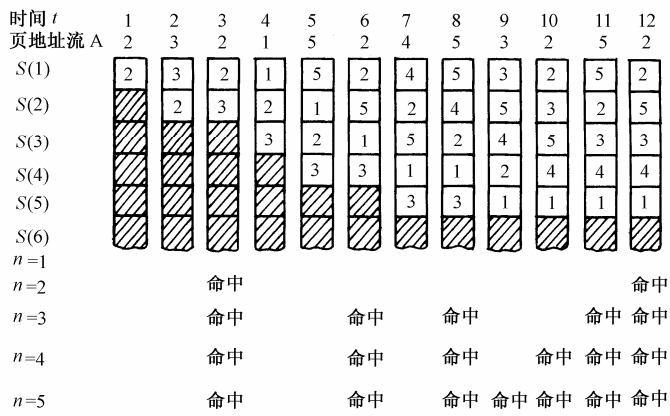


图 3-22 用堆栈处理技术对 LRU 的页地址流的分析

上述三种替换算法的特点如下：

- (1) FIFO。每次替换以先进入者为对象，是非堆栈型算法，实页数增加有时命中率反而降低，整体分析命中率较低。
- (2) LRU。每次替换以之前未命中最多者为对象，属堆栈型算法，实页数增加命中率上升，整体分析命中率较高，实用性强。
- (3) OPT。每次替换以之后最少使用者为对象，属堆栈型算法，实页数增加命中率上升，整体分析命中率最高，但实用困难，用于理论分析。

3.2.2 LRU替换算法的实现

1. 使用位法

这种方法用于“主-辅”层次的虚拟存储器。其基本思想是：给每个实页设一个使用位（访问位），开始时所有实页的使用位均为零；某个实页的任一个存储单元被访问，则由硬件在该页的使用位置“1”；在替换时，找使用位为零的实页即可。

前述的页表是用于每个程序（任务）的，而使用位是对应实存页的，所以不能把它放在页表内，而应另外建立一个实存页表，如表 3-3 所示。该表由操作系统内的“存储管理”负责记录实存中每页的状况。

表 3-3 实存页表

实 页 号	占 用 位	程 序 号	段 页 号	使 用 位	程序优先位	H_s
0						
1						
⋮						
15						
⋮						

实存页表中每一项的含义如下：

(1) 实页号。每实页按实页号 $0, 1, 2, \dots, n$ 顺序占用该表中一行。因为是顺序占用，故该表项可以略去，用相对于页表起点（起始地址）的相对位置（偏移量）来表示。

(2) 占用位。占用位为“1”表示该实页已占用（即有信息在内），为“0”表示该实页空闲。

(3) 程序号。占用该实页的程序（任务）编号。

(4) 段页号。占用该实页的程序（任务）的段号、页号。

(5) 使用位。前面已述。

(6) 程序优先位。本页程序的优先级。

(7) H_S 。也称历史位（或未使用过的计数器）。在确定的 Δt 时间间隔内（如 Δt 为几毫秒、几秒或几分），扫描所有使用位：若使用位为 0，则该页的 H_S+1 ，并让使用位仍为 0；若使用位为 1，则该页的 H_S 置 0，同时将使用位置 0。所以经过 Δt 后，实存页表内所有使用位均为 0，而 H_S 则记录了该页在近期内使用的情况（即在若干连续的 Δt 时间内，该页未被访问的历史情况）。

使用位法的替换过程如下：初始时，实存页表内全部空闲，所有的占用位、使用位、 H_S 均为 0。各个程序（任务）逐页调入，凡被占用的实存页，其占用位为 1，程序号、段页号、程序优先位置上相应值。一旦实存全部占用，所有的占用位为全 1。凡使用（访问）过的页，其使用位置 1。在发生页面失效时，就找使用位为 0 的页替换。如果在全部页均使用过（即使用位为全 1）的情况下发生页面失效，则无法判定哪一页应被替换。故使用位为全 1 的情况不允许出现。为此，一旦出现此种情况，就由硬件强制将全部使用位清 0。为了能在可替换的页中（即使用位为 0 的页）找出最久未使用的页，由硬件定时 Δt 扫描实存页表，对使用位、 H_S 作上述的处理（见 H_S 的说明）。在页面失效时，就寻找使用位为 0、而 H_S 数值最大者予以替换。替换后（即新页内容已置入），相应的使用位、 H_S 均清 0。

由于替换时主存与辅存间的页传送时间相当长，因此使用位和 H_S 的状态判断以及它们的处理等均用软、硬件结合的方法实现。

2. 堆栈法

LRU 是堆栈型替换算法（见前述）。堆栈法的基本思想是：按实存页面数组成有 2^n 行（项）的堆栈，栈顶为最近访问过的页的页号，栈底为近期最久没有访问过的页的页号，即被替换的页的页号。在访问主存时，把被访问的实页地址与堆栈中各行的 n_v 值相联比较，如果没有相符的（即未找到），则把该实页地址压入栈顶，堆栈所有项顺次下推一个位置；如果有相符的（即找到了），则把该实页地址压入栈顶，这相当于把堆栈中存放该页的 n_v 上移栈顶，而原来在 n_v 之上的各项下推一个位置。当堆栈全被装满（即实存全被占用），又出现页面失效时，栈底的项（即最久未使用的实页地址）就是替换对象，被压出堆栈。所以，堆栈法的特点是：栈单元的几何位置反映了被访问情况（即被访问过的先后次序）；栈单元的内容为实存的页号，因为这种堆栈要求有相联比较、既能全下推又能部分下推以及能由中间抽取一项（行）等功能，如用全硬件实现，只适用于容量较小的组相联的 LRU 替换。对于全相联的“主-辅”层次，则堆栈功能用软件实现。LRU 用堆栈法实现，见图 3-23。

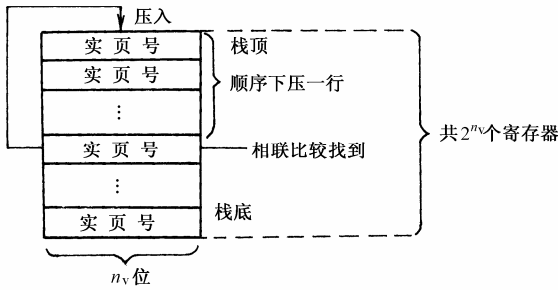


图 3-23 用堆栈法实现 LRU 替换算法

3. 比较对法

堆栈法需有相联比较功能，用硬件实现速度较低，价格较贵。而比较对法不用相联比较，而用一般的门电路、触发器等硬件实现 LRU 算法。它的基本思想是：将各页成对组合（形成比较对），用触发器状态表示每个比较对内的访问次序，再用“与门”求出 LRU 页（即近期最少访问的页）。下面举例说明其实现方法。例如，现有 A, B, C 三页，可组合成 $AB=BA$, $AC=CA$, $BC=CB$ 三对。使用三个触发器 T_{AB} , T_{AC} , T_{BC} ，分别表示 AB 对、AC 对、BC 对。若 $T_{AB}=1$ ，则表示 A 比 B 更近被用过（即 B 比 A 次序低，替换时优先）；若 $T_{AB}=0$ ，则表示 B 比 A “更近”被用过（即 A 比 B 次序低，替换时优先）。 T_{AC} , T_{BC} 同样定义。在这三对之间，如果 $T_{AB}=1$, $T_{AC}=1$, $T_{BC}=1$ ，则次序为 $A \rightarrow B$, $A \rightarrow C$, $B \rightarrow C$ ，综合次序为 $A \rightarrow B \rightarrow C$ 。如果 $T_{AB}=0$, $T_{AC}=1$, $T_{BC}=1$ ，则次序为 $B \rightarrow A$, $A \rightarrow C$, $B \rightarrow C$ ，综合次序为 $B \rightarrow A \rightarrow C$ 。上述两种情况均表示 C 为替换对象。上述关系用布尔代数式表示为

$$C_{LRU}=T_{AB} \cdot T_{AC} \cdot T_{BC}+\bar{T}_{AB} \cdot T_{AC} \cdot T_{BC}=T_{AC} \cdot T_{BC}$$

同理可得

$$B_{LRU}=T_{AB} \cdot \bar{T}_{BC}$$

$$A_{LRU}=\bar{T}_{AB} \cdot \bar{T}_{AC}$$

用硬件实现本例的比较对法的线路如图 3-24 所示。 A_{LRU} , B_{LRU} , C_{LRU} 为三个输出，哪个为 1 则表示它是替换对象。在每次访问存储器（即实存页）后，需改变比较对触发器的状态。访问 A 页后，使 $T_{AB}=T_{AC}=1$ ，表明 A 比 B 近，A 比 C 近；访问 B 页后，使 $T_{AB}=0$, $T_{BC}=1$ ，表明 B 比 A 近，B 比 C 近；访问 C 页后，使 $T_{AC}=0$, $T_{BC}=0$ ，表明 C 比 A 近，C 比 B 近。由此可得到触发器输入控制逻辑，如图 3-24 所示。

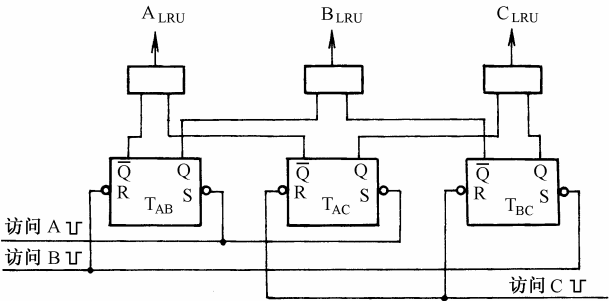


图 3-24 用比较对法实现三页 LRU 替换算法

当页数增加时，比较对法所用硬件成本迅速增加。每页需一个“与门”，则 P 页需 P 个

“与门”；每个“与门”的输入端数为每页有关的对数，即页数减 1， P 页时每个“与门”的输入端数为 $P-1$ ；比较对数为 $P(P-1)$ ，由于其中一半是重复的，所以比较对数为 $P(P-1)/2$ ，触发器数即是比较对数。页数、比较对数、门数、门输入端数关系如表 3-4 所示。

表 3-4 比较对法使用硬件数

页 数	3	4	8	16	64	256	...	P
比较对数 (触发器数)	3	6	28	120	2016	32640	...	$\frac{P(P-1)}{2}$
门 数	3	4	8	16	64	256	...	P
门输入端数	2	3	7	15	63	255	...	P

因为触发器个数随页数迅速增加，所以比较对法只适用于采用组相联映像的“主-Cache”层次。在页数少时，它比堆栈法或使用对法易于实现。若组内页数大于 8，则不可行，这时可采用多级状态位（比较对）技术以减少所用位数。例如， $P=16$ ，可分成群、对、行三级。先分 4 群，每群 2 对，每对 2 行，每行对应 1 页，则 $4 \times 2 \times 2 = 16$ 页。根据上表可知，4 群需要 6 个状态位（比较对）；2 对需要 1 个状态位，4 群共有 $2 \times 4 = 8$ 对，则需要 4 个状态位；2 行（页）需要 1 个状态位，8 对共有 $2 \times 8 = 16$ 行，则需要 8 个状态位。故状态位（比较对数或触发器数）总数 = 6（选群）+ 4（选对）+ 8（选行）= 18。这比单级时 120 个少得多。

综上所述，设计替换算法应依据下列两点：

（1）如何对每次访问进行记录，上述三种方法所用的记录方式均不相同。

（2）如何根据记录信息判定替换对象。实现方法和所用映像方法密切相关。对于“主-辅”层次全相联映像宜采用使用位法，对于“Cache-主”层次组相联映像宜采用比较对法或堆栈法。替换算法的设计和实现密切相关，随着元器件和相联存储器的改进，必然会出现更好的实现方法。

3.3 并行主存系统

为了提高主存储器的吞吐率，可以采取多种措施，其一是增加一次访问主存读出的信息量，从一个单元增加到多个单元。这就要将存储器分成多个模块，可以并行地读出多个单元，这种存储器结构就是并行存储器（Parallel Memory）。

3.3.1 并行主存系统频宽分析

图 3-25 是一个字长为 W 的单体主存，一次可以访问一个存储器字，所以主存频宽 $B_M = W/T_M$ 。若 W 与 CPU 字长一致，则 CPU 在一个存取周期 T_M 时，获得信息量的速率就为 B_M 。这种主存称为单体单字存储器。

要提高主存频宽 B_M ，使之与 CPU 速度匹配，在相同芯片条件下（即相同的 T_M 时），只有提高存储器字长 W ，图 3-26 是在一个 T_M 内读取 4 个字的单体多字存储器， $B_M = 4 \times W/T_M$ ，其速率是单体单字存储器的 4 倍。

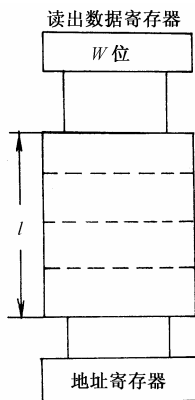
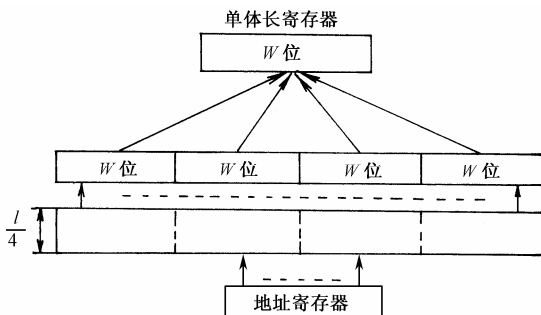


图 3-25 单体单字存储器

图 3-26 单体多字 ($m=4$) 存储器

大容量半导体主存往往由许多相同的存储芯片组成, 因此可形成单体多字、多体单字和多体多字的交叉存取主存系统, 该系统称为并行主存系统。模 m 在单体多字方式中为一个主存单元所包含的字数, 在多体单字中为分体体数。主存采用多体单字方式组成时, 其实际频宽比单体多字高。增大 m 的值, 可以提高主存系统的频宽 B_M , 但 B_M 的值并不随 m 值增大而线性提高。首先, 模 m 越大, 存储器数据总线越长, 总线上并联的存储芯片越多, 使传输延时增大。其次, 存在系统效率问题, 如对模 m 交叉, 如果都是顺序地取指令, 效率可提高 m 倍; 如果出现转移, 效率就会下降, 转移频度越高, 下降越明显, 而数据的顺序性比指令差, 实际频宽 B_M 可能更低一些。

现在, 通过一个有 m 个独立分体的并行主存系统, 分析其 B_M 与地址流的关系。设 CPU 送出的地址流为 A_1, A_2, \dots, A_g , 在每个 T_m 之前, 对地址队列扫描, 截取 A_1, A_2, \dots, A_K , 称为申请序列, 其 K 个地址中, 不会发生分体冲突 (即不会有两个以上地址访问同一个分体)。 K 为一个随机变量, 最大值为 m , 由于会发生分体冲突, 往往 $K \leq m$ 。 K 个地址, 可同时访问 K 个分体, 因此系统效率取决于 K 的平均值。 K 越接近 m , 效率就越高。

设 $P(K)$ 是 K 的概率密度函数, 其中, $K=1, 2, \dots, m$ 。即 $P(1)$ 是 $K=1$ 的概率。 $P(2)$ 是 $K=2$ 的概率, $P(m)$ 是 $K=m$ 的概率。 K 的平均值用 B 表示, 则

$$B = \sum_{K=1}^m KP(K) \quad (1)$$

B 实际上是每个 T_M 所能访问到的平均单元 (字) 数, 它正比于主存频宽 B_M (只差一个常数比值 W/T_M)。 $P(K)$ 与程序密切相关, 如访存都是取指, 影响最大的是转移概率 λ , 即当前指令的下一条指令地址为非顺序地址的概率。在访存地址序列 A_1, A_2, \dots, A_K 时, 一旦出现成功转移, 则转移指令之后, 同时取出的其他指令就无效了。当申请访存队列中 A_1 是转移指令并且转移成功时, 则并行读出的其他 $m-1$ 条指令均无效 (设 $K=m$), 相当于 $K=1$, 所以 $P(1)=\lambda$; $K=2$ 的概率是 A_1 不是转移指令 (其概率为 $1-\lambda$), A_2 是转移指令并且转移成功, 所以 $P(2)=[1-P(1)]\lambda=(1-\lambda)\lambda$ 。以此类推, $P(3)=[1-P(1)-P(2)]\lambda=((1-\lambda)^2)\lambda$, 则

$$P(K)=(1-\lambda)^{K-1}\lambda \quad (1 \leq K \leq m)$$

如果前 $m-1$ 条指令均不转移, 则不管第 m 条指令是否转移, $K=m$, 故 $P(m)=(1-\lambda)^{m-1}$ 。

将 $P(1), P(2), \dots, P(m)$ 代入 (1) 式

$$B=1\times\lambda+2\times(1-\lambda)\times\lambda+3\times(1-\lambda)^2\times\lambda+\cdots+(m-1)(1-\lambda)^{m-2}\times\lambda+m\times(1-\lambda)^{m-1}$$

用数学归纳法化简，可得

$$B=\sum_{\lambda=0}^{m-1}(1-\lambda)^i$$

这是一个等比级数，因此

$$B=\frac{1-(1-\lambda)^m}{\lambda}$$

由此可见，若每条指令都是转移指令并转移成功， $\lambda=1$ ， $B=1$ ，则并行多体交叉存取的实际频宽降低到与单体单字一样。若所有指令都不是转移指令， $\lambda=0$ ，则 $B=m$ ，此时多体交叉并行主存的效率最高。

3.3.2 单体多字存储器

并行主存系统有两种组成方式：单体多字方式和多体并行方式。虽然每个半导体存储芯片内部已经有了译码、读放电路，但在多存储器组成并行存储系统时，仍需外加地址译码电路，用于地址锁存和片选等功能。当并行的存储器共用一套地址寄存器和译码电路时称为单体方式，如图 3-27 所示。多个并行存储器与同一个地址寄存器连接，同时被一个单元地址驱动，一次访问读出的是沿 n 个存储器顺序排列的 n 个字，故也称单体多字方式。若 n 个并行工作的存储器具有各自的地址寄存器和地址译码、驱动、读放、时序电路，能各自以同等的方式与 CPU 传递信息，形成可以同时工作又独立编址且容量相同的 n 个分存储体，这就是多体方式，如图 3-28 所示。多体主存系统能够实现多个分体并行存取，一次访问并行读出的 n 个字不像单体方式那样一定是沿存储器顺序排列的存储单元内容，而是分别由各分体的地址寄存器指示的存储单元信息。多体方式与单体方式相比控制线路较复杂，但其地址设置灵活，已被大多数大中型计算机所采用。

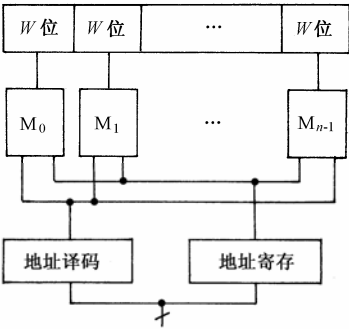


图 3-27 单体多字并行主存系统

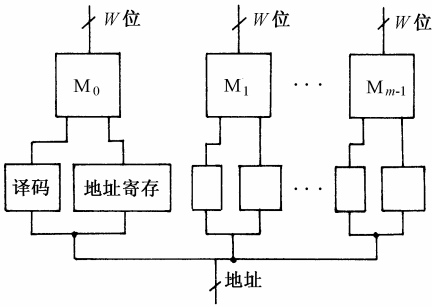


图 3-28 多体并行主存系统

3.3.3 多体交叉存储器

1. 多体交叉编址

并行主存系统指包含多个能够独立编址的多体并行主存系统，各分体间的地址编号采用交叉方式。现以具有 4 个分存储体的主存为例说明常用的编址方法。4 个分体 M_0, M_1, M_2, M_3 的编址序列如图 3-29 所示。框内序号表示存储单元的地址编号 $j=0, 1, 2 \cdots$ 。

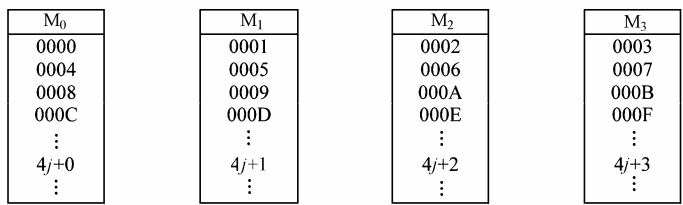


图 3-29 4 个分体编址序列

地址编号规则如下：

- ① 地址按并行分体横向顺序编号。地址序号连续的两个存储单元依次分布在相邻两个存储分体中，而不是在一个体内排序，故称多体交叉编址。
- ② 同一分体内相邻存储单元地址编号相差 4。
- ③ M₀ 分体每个单元地址的二进制编码最后两位都是 00，M₁ 分体地址最后两位均是 01，M₂ 分体最后两位二进制地址是 10，而 M₃ 分体最后两位地址均是 11。任何一个存储单元二进制地址编号的末两位正好指示该单元所属分体的序号，这两位就称为内存体号。访问主存时只需判断地址的体号就能决定访问的是哪个分存储体了。
- ④ 同一分体内每一个单元地址除去体号后的高位地址码称为体内地址，它正好是体内单元的顺序号。由体内地址就能决定访问单元在分体里的位置。

地址交叉排列的目的是为了便于分体同时工作。假设有 4 条顺序执行的指令，一条指令一个字长，地址编码是 0，1，A，B，分配在不同存储器上。并行主存工作时，对于单体的并行系统，一个存储周期 T_M 只能读取 4 个地址连续的存储单元内容，比如 0~3 号单元内容，只获得 0，1 单元内两条有用指令，实际频宽下降；对于多体交叉存储器，由于 4 条指令分属于 4 个分体，4 个地址寄存器可以指示 4 个不连续的地址，这 4 条指令能够在一个存储周期里读出。可见，多体和单体并行主存系统虽然最大频宽可以相同，但多体地址可以灵活设置，只要是在不同分体中，就能得到比单体更高的频宽。因为程序常有转移出现，使得实际访问地址不一定均匀分布在交叉分体上，致使效率下降。统计表明，采用多体并行主存结构的计算机系统获得的实际频宽约是最大理想频宽的 1/3。此外，工艺设计不完善时也会引起传输线延迟增加而降低实际存取速度，所以实际效率与理想效率尚有一定差距。

2. 多体交叉存储体分时工作原理

主存与 CPU 交换信息的通道只有一字的宽度，为了在一个存储周期里访问 n 个信息字，在多体并行主存系统中采用了分时工作的方法，目前普遍采用的是分时读出法。现设多体交叉存储器由 4 个分体组成，每个存储体一次读/写一个字。分体分时启动，即每隔 $1/4$ 存储周期启动一个分体，见图 3-30，M₀ 体在第一个主存周期开始读/写，经过 $1/4T_M$ 启动 M₁ 体，M₂ 和 M₃ 体分别在 $1/2T_M$ ， $3/4T_M$ 时刻开始它们各自的读/写操作，4 个分体以 $1/4T_M$ 的时间间隔进入并行工作状态。假定 $T_M=2\mu s$ ，普通存储器只能读/写一个字，4 体并行工作时， $2\mu s$ 内 CPU 依次发出 4 个读/写命令，访问 4 个字，对 CPU

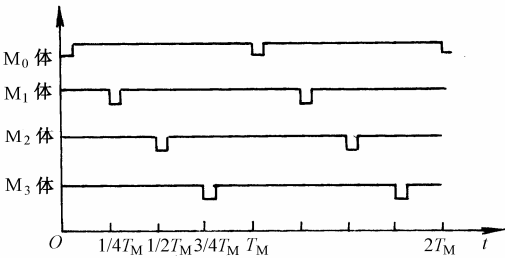


图 3-30 4 体并行分时工作

来说，相当于每 $0.5\mu\text{s}$ 读取一个字，虽然每个存储体仍以 $2\mu\text{s}$ 速度存取。这对主存系统来说，仿佛有一串地址流以 $1/4$ 存储周期速度流入，便有一串信息流以同样速度流出主存系统，这种类似流水的工作过程很适于与流水式中央处理机的联系。

另一种并行存取的处理方法是同时启动 4 个分体，使 4 个分体在一个存储周期 T_M 内同时被访问，一次读出 4 个字，然后以一定顺序分时使用总线实现对外传送。

3. 多体交叉存储器组成

多体交叉存储器组成框图如图 3-31 所示，主要有存储体、存储器控制器（简称存控）和总线控制三部分。存控用于组织多体并行工作，实现分时读出的工作方式，管理信息流动次序和流动方向。当 CPU 或通过 IOP 的外设向主存系统重叠发出访问要求时，存控首先对这些访问源进行排队，设计人员事先已经根据所有访问源的性质区分轻重缓急，排出优先级别。比如高速通道传送实时性很强的信息；突发故障时需要紧急处理的信息往往优先级别高，排在优先访问的位置。存控选择其中优先级别最高的先访问，并向它所访问的分体发出启动信号以及访问地址。如果被访问的那个存储分体正处于工作状态，无法接受访问，则暂时取消该访问源的排队资格，让给优先级别稍低的访问源访问其他存储体。每个存储分体不但有自己的读/写控制线路、数据缓存设备，而且各具“忙闲”状态触发器。当存储分体接收到存控的启动命令时，如果“忙闲”状态触发器处于 0 态，表示空闲，分体按存控命令操作，访问时忙闲状态触发器置 1 直到存储体一个读/写操作完毕重新置 0；若忙闲触发器正处于 1，则不接收新的启动命令，存控通过检测忙闲触发器的状态控制总线上的信息流向。

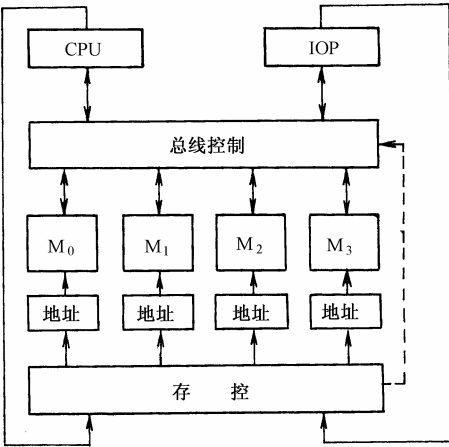


图 3-31 多体交叉存储器组成

目前越来越多的大中型机采用多体并行与单体多字相结合的并行主存系统，每个分体的宽度不是单字而是多字，进一步提高了频宽。例如有的机器 4 个分体并行，每个分体一次读两个字，每个字长为 4 字节，虽然存储周期为 $2\mu\text{s}$ ，但其最大频宽可达 $16\text{B}/\mu\text{s}$ 。

3.3.4 并行主存系统

1. 地址空间的划分

并行存储器是将主存划分成多个相同容量的存储模块，各有自己的地址寄存器和数据寄存器，在同一时间允许对多个模块独立地进行访问。划分地址空间有三种模式：按高位地址

划分、按低位地址划分、混合划分。

按高位地址划分如图 3-32 所示。若主存地址的高位地址字段有 n 位，则可将主存划分成 2^n 个模块。例如，高位地址字段有 2 位，则可将主存划分成 $2^2=4$ 个模块；有 3 位，则可将主存划分成 $2^3=8$ 个模块。访问存储器时，高位地址字段经译码选择存储器模块，低位地址字段送地址寄存器，指向相应模块的某一单元。这种划分方法适合于把程序和数据分开放在不同的模块内，对于重叠执行的指令可以避免取指令和取操作数的时间冲突。也可以把不同用户的程序和数据放在不同的模块内，由处理机分时（按时间片轮流）操作。这种划分方法和用法实现简单，但不能充分发挥并行存储器提高信息吞吐率的优点，且失去了多用户共享主存的特点。当然，可将某些模块指定为外设数据缓冲区，使外围设备和处理机并行工作。但这些用法都有较大局限性，不能明显地提高主存的速度（吞吐率），所以按高位地址划分实际采用较少。

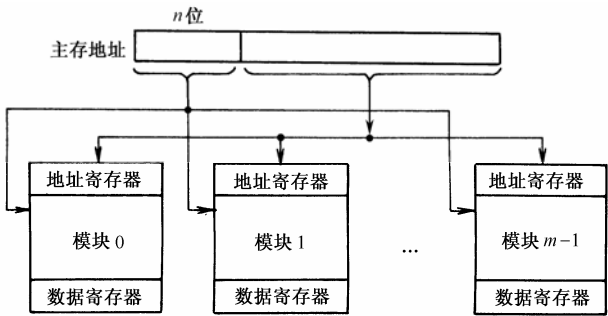


图 3-32 并行存储器按高位地址划分模块

按低位地址划分如图 3-33 所示。与前述方法相反，地址码的低位字段经过译码选择不同的存储器模块，而高位字段指向相应模块内部的存储单元。这样，相连地址分布在相邻的不同模块内，而同一模块内的地址都是不相连的。这种并行存储器结构又称存储器交叉。

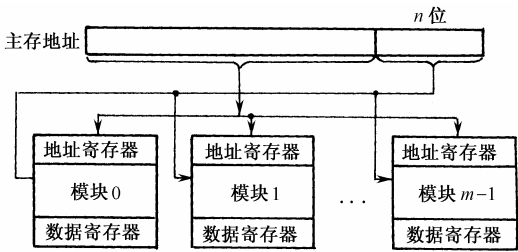


图 3-33 并行存储器按低位地址划分模块

如果程序段和数据块（数组）按连续地址在主存中存储和读取，则采用交叉存储器结构最适于实现多模块并行访问，可以成 10 倍地提高主存的有效访问速度。但是，遇到程序转移或个别数据存取时，地址不一定均匀分布在多个交叉存储器模块内，模块的使用率会降低。所以， m 个交叉模块的使用率是一个随机值，大致在 $1\sim m$ 。如果所有的访问均是随机性的，用单服务、先来先服务排队论模型进行模拟，可求得 m 的取值和主存使用率 B （字/主存周期）之间关系，如表 3-5 所示。

表 3-5

<i>m</i>	1	2	4	8	16
<i>B</i>	1.0	1.5	2.219	3.245	4.704

可以看出，随着 m 的增加， B 以 \sqrt{m} 增大（或更准确地用 $m^{0.56}$ 表示）。由于指令流和数据流不会全是随机性的，因此 B 值总会比 \sqrt{m} 值大。一般认为，在通常情况下，采用 $m=16$ 意义已不太大，大多取 $m=2\sim 8$ 。如果主存访问速度仍然满足不了需要，那就用“Cache-主存”层次，不能单纯靠增大 m 值来解决问题。当然，有 Cache 时，仍可采用并行存储器结构，以便能快速向 Cache 传送信息。

交叉存储器由多个模块组成连续的地址空间，这带来了三个问题：一是模块数必为 2^n ；二是任一模块失效会造成地址空间缺陷而导致整个程序无法运行；三是不利于存储器模块数增量式扩展，即不能以一个模块作为增减主存容量的最小单位。而按高位地址划分模块就不存在这三个问题。若以加强存储器配置的灵活性为目的，则可采用按高位地址划分模块。

上述两种方法的结合就是混合划分。即先按高位地址分成几个模块，而模块内部又按低位地址交叉。

2. 访问周期的控制

并行存储器有两种访问方式：同时访问和交叉访问。同时访问的基本思想是：所有模块在一次访问周期内同时启动，相对各自的数据寄存器并行地读出或写入信息。同时访问的并行存储器能一次提供多个数据或多条指令，适用于对多数据流或多指令流进行并行处理。它要求多处理机接收或提供信息的频宽与并行存储器同时读/写的频宽相匹配。因此，处理机与存储器模块之间的互连方式起着很大作用。例如图 3-34 所示为交叉开关的互连网络，它是在纵向和横向两套多总线所有交叉点上设置开关，能以任意排列模式实现存储器模块 M_0, M_1, \dots, M_{m-1} 与多个处理机 $CPU_0, CPU_1, \dots, CPU_{n-1}$ （包括 I/O 处理机 $IOP_0, IOP_1, \dots, IOP_{s-1}$ ）之间的同时连接。交叉访问的基本思想是： m 个模块按一定顺序轮流启动各自的访问周期，启动两个相连模块的最小时间间隔等于单模块访问周期的 $1/m$ 。交叉访问的并行存储器的工作时间图如图 3-35 所示。就每一个存储器模块而言，它的连续两次访问的时间间隔仍等于单模块访问周期。所以交叉访问存储器的有效总频宽与同时访问存储器是一样的，但是各模块错开启动时间，并行存储器提供或要求数据的频宽在各模块之间分布得更均匀了，从而降低了对互连网络的要求。使用最普遍的互连方式是分时总线，如图 3-36 所示。总线时间分割成时间间隔相等的若干时间片，每个存储器模块只在分配给它的时间片内占用总线，与中央处理器交换信息。交叉访问并行存储器的另一个优点是，它最适合流水线结构。它既能以流水线方式向 CPU 流水线操作部件提供数据，又能以流水线方式从流水线操作部件接收结果，双方能取得较理想的数据流频宽匹配。

为了配合并行存储器工作，在 CPU 内设置指令缓冲部件和数据缓冲部件，也可在 CPU 和主存间设立 Cache。同时，对一次读出的多条指令进行“先行控制”技术处理以加快其执行过程，即提前检查后续指令，提前将它们所需的数据从主存调出。一次调出一个数据块，连续使用，就能充分发挥并行存储器提高处理速度的能力。

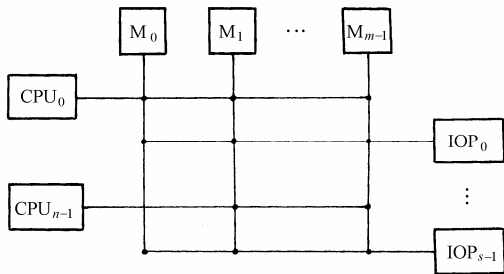


图 3-34 交叉开关互连网络

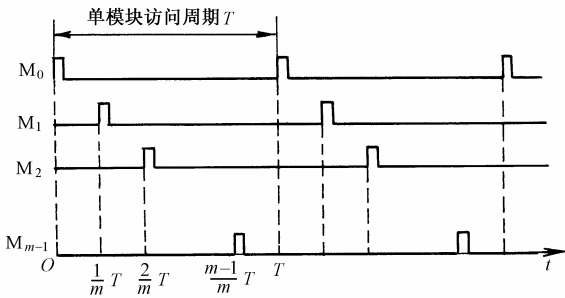


图 3-35 交叉访问并行存储器工作时间图

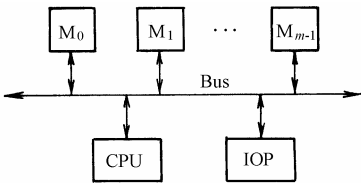


图 3-36 分时总线的互连方式

3.4 高速缓冲存储器（Cache）

在本章的 3.1 节已提出，为了弥补主存速度不足，在存储体系结构中出现了“Cache-主存”层次，从 CPU 观察，它具有 Cache 的速度和主存的容量。Cache 在当代计算机系统中已普遍使用，成为提高系统性能的不可缺少的功能部件。Cache 的容量不断增大，Cache 的管理实现了全硬化，Cache 的部件已高度集成，这些已成为当代 Cache 的特征。本节从原理着手，以系统结构的观点，着重介绍 Cache 的工作原理、多机系统的 Cache 结构和影响 Cache 性能的因素。

3.4.1 Cache基本结构和工作原理

1. 访问局部化

对大量典型程序的运行情况的分析结果表明：在一个较短的时间间隔内，程序所产生的访存地址往往集中在存储器地址空间的小范围内。指令地址分布基本上是连续的，循环程序段和子程序段要反复多次执行，因此，对此类地址的访问就自然具有时间上集中分布的倾向。数据分布的集中倾向不如指令明显。但对数组的存储和访问以及工作单元的选择可使其地址相对集中。这种对局部范围的存储器地址频繁访问，而对此范围外的地址区域访问甚少的现象称为程序访问局部化性质。

在时间间隔 $(t-T, t)$ 内被访问的信息集合用 $W(t, T)$ 表示，也称为工作集合。根据程序访问局部化性质， $W(t, T)$ 随时间的变化是相当缓慢的。把这个集合从主存中移至（读出）一个能高速访问的小容量存储器内，供程序在一段时间内随时访问，可大大减少程序访问主存的次数，从而加速程序的运行。这个介于主存和 CPU 之间的高速小容量存储器就称为 Cache。

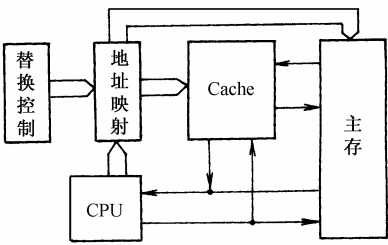


图 3-37 Cache 存储系统原理框图

“块”相似于“主存-辅存”层次中的“页”。主存地址通过“主存-Cache”地址映像变换机构判定该字所在块是否已在 Cache 中。如在，则主存地址变换成 Cache 地址，访问 Cache；如不在，则发生 Cache 块失效（Cache 不命中），需访问主存，且将包含该字的一块信息装入 Cache。若 Cache 已满，则按某种替换策略，把该块替换进 Cache。

所以，程序访问局部化性质是 Cache 得以实现的原理基础，而高速（能与 CPU 匹配）则是 Cache 得以生存的性能基础。

Cache 介于 CPU 和主存之间，它的工作速度数倍于主存，全部功能由硬件实现，并且对程序员是透明的。Cache 存储系统的原理框图见图 3-37。

2. Cache基本结构

Cache 基本结构如图 3-38 所示。Cache 和主存都分成块（行），每块（行）由若干个字（字节）组成，

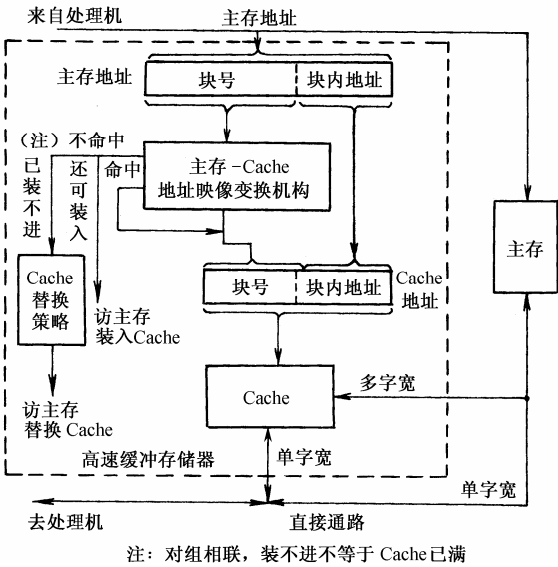


图 3-38 Cache 基本结构

Cache 的速度一般为主存的 2~4 倍，而访问 Cache 中的指令和数据的时间只有一般访问主存时间的 25%~50%。因此，只要 Cache 命中率足够高，就相当于能按 Cache 的速度来访问大容量的主存。

Cache 的设计要求是：在价格允许的前提下，提高命中率和缩短访问时间，尽可能减少因不命中造成的时间延迟以及尽可能减少为修改主存所花的时间开销。

为了使 Cache 能与 CPU 在速度上相匹配，Cache 一般采用与 CPU 相同的半导体工艺所制成的大规模集成电路芯片。为了更好地发挥 Cache 的高速性，在物理位置上，使 Cache 尽量靠近处理机或就在处理机中，而不放在主存模块中，在 VLSI 得到迅速发展的今天，高档微处理器芯片中（如 32 位的微处理器芯片 Intel Pentium，Motorola 68030）不仅集成了存储管理部件，而且集成了一定容量的 Cache。目前，一般用高速 SRAM（静态 RAM）芯片组成

相当容量、价格适宜的 Cache。如再配上以合适的调度算法为基础、全部硬化的地址映像和变换部件,就能实现高主振频率的 CPU 的零等待(可使 CPU 访问存储器在其时序中不插入等待状态 T_w)。

在 Cache 发生块失效时,由于主存调块的时间是微秒级,不能在此时采用切换任务(即程序换道)方式来减少 CPU 等待时间,所以,除了 Cache 到 CPU 的通路外,在主存和 CPU 之间还有直接通路,如图 3-38 所示。这样在 Cache 块失效时,就不必等主存把所需单元所在块调入 Cache 后,再由 CPU 对其进行读取;而是使 Cache 调块与 CPU 访问主存同时进行,这就是通过直接通路实现读直达;同样地,也可实现 CPU 直接写入主存的写直达。故 Cache 既是“Cache-主存”层次中的一级,又是 CPU 与主存间的一个旁视存储器。

为了加快调块,在主存与 Cache 之间采用主存多体交叉多字结构(见前述并行存储器),每块的容量一般等于一个主存读/写周期内由主存所能访问到的字数。例如,巨型机 CRAY-1 的主存是模 16 交叉,每个分体是单个字,所以其存放指令的 Cache 的块容量为 16 个字,在一个主存周期内即可完成指令块调入 Cache。

由于主存被计算机系统的多个部件共用,难免发生访存的竞争,故应把 Cache 访问主存的优先级尽量提高,一般要高于通道访存的级别,因为 Cache 调块时间只占用 1~2 个主存周期,这样安排不会对外设访存带来太大的影响。

3. Cache 的读/写操作

CPU 进行读存储器操作时,根据其送出的主存地址分为两种不同情况:一种是需要的信息已在 Cache 中,那么直接访问 Cache 就行了;另一种是所需信息不在 Cache 中,就要把该单元所在的块从主存调入 Cache。后一种情况又有两种实现方法:一种是将块调入 Cache 后再读入 CPU;另一种是前述的读直达(也称通过式读或通过式加载),后一种方法得到普遍使用。在调入新块时,如果 Cache 已占满,由替换控制部件按已定的替换算法实现替换。

Cache 中的块是主存中相应块的副本。如果程序执行过程中要对某块的某单元进行写操作,有两种方法:一种称为标志交换方式,即只向 Cache 写入,并用标志注明,直至该块在替换中被排挤出来,才将该块写回主存,代替未经修改的原本;另一种称为写直达(也称通过式写或通过式存),即在写入 Cache 的同时,也写入主存,使原本和副本同时修改。前一种方法写操作速度较快,但在该块写回主存之前,主存中的原本由于未及时修改而失去时效。后一种方法实现简单,且能随时保持原本的时效,这对于多处理机共享一个原本特别重要。另外,还有一种写操作情况:当被修改的单元不在 Cache 时,写操作可直接对主存进行,而不必把包含该单元的块调入 Cache 后再修改,因为程序访问局部化对写操作不很明显。

3.4.2 Cache 的替换算法分析

Cache、主存间的地址映像和变换以及替换算法,原理上与“主存-辅存”层次相同(见前述),虚地址指的是主存地址,实地址指的是 Cache 地址。Cache 与主存的访问时间比为 $1/2 \sim 1/4$,因此地址变换用硬件实现(如硬化快表),替换算法也是采用堆栈法或比较对法等硬化处理。Cache 地址变换一般采用组相联映像,如采用全相联映像,将使变换表容量过大而降低访问速度。

从图 3-38 可见,从送入主存地址到访问 Cache,包含地址变换和访问 Cache 两部分时间。如果使两部分时间基本一致,就可以采用流水方法。对 Cache 的一次访问,需有下列过程:

多个请求源同时提出访问 Cache 时，首先要经过优先级排队；访问目录表进行地址变换；访问 Cache；从 Cache 中选择所需的单元；修改 Cache 中块（行）使用状态标志，等等。若每一个步骤需时间 t ，则 n 个步骤共需时 nt 。采用流水处理后，虽然每次访问 Cache 所需时间仍为 nt ，但多条指令或数据一旦进入该流水线，可在 t 时间内得到一个访问 Cache 的结果，从而大大提高了 Cache 的吞吐率。

绝大多数 Cache 采用 LRU 替换算法，Cache 出现块失效时，一般采用“按需取进法”，即在 Cache 不命中时，才将要访问的单元所在的块（行）调入 Cache。由于 Cache 的命中率对机器的速度影响很大，若辅之以在未用到之前就预取进 Cache 的预取算法，则会进一步提高命中率。为便于硬化实现，通常只预取直接顺序的下一块（行），即在访问到主存的第 i 块（行）时（不论是否已取入 Cache），第 $i+1$ 块（行）才是可能的预取块（行）。至于何时取进，有恒预取或不命中时预取等方法。恒预取是指：只要访问到按主存编号第 i 块的某单元，不论是否命中，恒发预取命令。不命中时预取是指：只有当访问第 i 块不命中时，才发预取命令。

采用预取法并非一定能提高命中率，它和很多因素有关，主要有下列两个因素：

（1）块（行）的大小的影响。如每块的字节数过小，则预取效果不大。但若每块的字节数过大，则会预取进不需要的信息，同时，可能把正在使用的或近期内就要用到的信息挤出去，反而会降低命中率。从已有模拟情况来看，若每块字节数超过 256，就会出现这种情况。

（2）预取访存的影响。要预取就有访主存预取块的开销和将它取入 Cache 的开销，以及将被替换的块由 Cache 送回主存的开销。这些开销增加了主存和 Cache 的负担，会干扰和延缓程序的执行。

所以，采用预取法的效果不能只从提高命中率来衡量，还要从按需取进去的不命中开销与预取法的不命中开销加预取开销（包括访存开销和对 Cache 的干扰）之和来比较。

设 T_C 为不命中时由主存调一块进 Cache 的时间，则 $T_C \times \text{不命中率}$ 为不命中开销。

设预取率 = 预取总块（行）数 / 访主存总块（行）数， P_a = 预取访主存和访 Cache 的开销，则 $P_a \times \text{预取率}$ 是预取法的预取开销。设

$$\begin{aligned} \text{访问率} &= \text{访 Cache 总次数} / \text{程序访 Cache 次数} \\ &= (\text{程序访 Cache 次数} + \text{预取访 Cache 次数}) / \text{程序访 Cache 次数} \\ A_C &= \text{预取干扰开销} \end{aligned}$$

它是由于预取而延迟、干扰了程序对 Cache 的访问所用的时间。则 $A_C \times (\text{访问率} - 1)$ 反映了预取法对程序访问 Cache 的影响。

这样，预取法只有在满足

$T_C \times \text{不命中率}(\text{按需取进法}) > [T_C \times \text{不命中率}(\text{预取法}) + P_a \times \text{预取率} + A_C \times (\text{访问率} - 1)]$ 才是可取的。在 Cache 和主存分别设置预取专用缓冲器，使预取访问 Cache 和访问主存均在主存、Cache 空闲时进行，这是减小预取干扰的好办法。

模拟结果表明，恒预取法使不命中率降低 75%~80%，不命中时预取法使不命中率降低 30%~40%，但前者所引起的 Cache、主存间信息传输量的增加比后者大得多，即预取开销前者比后者大得多。

3.4.3 Cache的透明性

由于 Cache 的地址变换和块替换算法的实现均依靠硬件,故“Cache-主存”层次对系统程序员和用户都是透明的,且 Cache 对 CPU 与主存间的信息通信也是透明的。对于 Cache 的透明性可能引发的问题及其影响需要慎重对待。

虽然 Cache 内存储的信息是主存内存储的部分信息的副本,但是主存内某单元的内容和 Cache 内对应单元的内容却可能在一段时间内是不同的。例如,CPU 修改 Cache 内容时,主存对应部分内容还没有变化;或者 I/O 处理机(IOP)已将新的内容输入主存某区域,而 Cache 对应部分内容却可能还是原来的。这些在通信过程中引起的 Cache 内容跟不上主存对应内容的变化,或者主存内容跟不上 Cache 内容的变化,在 Cache 对处理机和主存均是透明的前提下,可能引发错误。

当处理机执行写入操作时,若只写入 Cache,则主存中对应部分仍是原来的,会对 Cache 的块替换产生影响,而且当 CPU、IOP 和其他处理机经主存交换信息时,会造成错误。为了解决这个问题,人们提出了主存修改算法。一般可用两种方法修改主存:写回法和写直达法。写回法即前述的标志交换方式(见 Cache 的读/写操作)。二者的区别是:前者仅在被替换时才将修改过的 Cache 块写回主存;后者在修改 Cache 的同时也直接写入主存。显然,写回法是把开销花在替换时,而写直达法则是花在每次写 Cache 时都要附加一个写主存的开销,而写主存所花费的时间比写 Cache 大多得。据对典型程序的统计表明,在所有的访存中约有 10%~36%是写操作。虽然写回法要写回整个块(行),而不是仅仅写回一个或两个单元,但写回法使主存的通信量比写直达法要小得多。例如,设 Cache 不命中率为 3%,块(行)的大小为 32 字节,主存模块数据宽度为 8 字节,写操作占有所有访存的 16%,且所有 Cache 块的 30%需要写回操作,则写主存次数占总的访主存次数的百分比是:写直达法为 16%,而写回法仅为 3.6% ($0.03 \times 0.30 \times 32/8$)。CPU 的不少写入操作是对程序某个中间结果的暂存。写回法有利于省去把中间结果写入主存的无谓开销,但在该块替换前,依然存在主存内容与 Cache 内容的不一致性。对于有 CPU、IOP 系统或多处理机的系统,需采用共享 Cache 或复杂的目录表系统,以解决处理机间经主存的信息通信问题。

写回法需设置修改标志用以确定是否要写回以及控制先写回、后读入的块操作顺序,从而使 Cache 复杂化。

在出现写不命中时,这两种方法都面临着一个在写时是否取的问题。它有两种解决方法:一种是“不按写分配法”,即当 Cache 写不命中时只写入主存,该单元所在块不从主存调入 Cache;另一种是“按写分配法”,即当 Cache 写不命中时除写入主存外,还把该单元所在块由主存调入 Cache。这两种策略对不同的主存修改算法其效果不同,但差别不大。写回法一般采用“按写分配法”,写直达法一般采用“不按写分配法”。

写回法和写直达法硬化实现时都需少量缓冲寄存器。缓冲器内寄存要写入的数据及要写入的单元的目标地址。缓冲器对 Cache 与主存是“透明”的,在设计时要处理好由它可能引起的错误(如另一个处理机要访问的主存单元的内容仍在缓冲器中)。写回法的缓冲器用于暂存将写回主存的块,不必等待被替换块写入主存后才能进行 Cache 取(即从主存将新块调入 Cache)。写直达法的缓冲器用于暂存写入的内容,由于 Cache 的速度比主存快,写入 Cache 完成后,CPU 不必等待写主存完成就能依靠 Cache 继续往下运行程序。

从可靠性上讲,写直达法比写回法要好。对前者,当 Cache 出错时,可依靠主存进行纠

正，因此，Cache 中只需要一位奇偶校验位。对后者，由于有效的块（行）只在 Cache 中，因此在 Cache 内需采用纠错码，增加冗余信息位来提高其内容的可靠性。从实现成本上讲，写回法比写直达法要低得多。写直达法需要较多的缓冲器和大量其他辅助逻辑来减少 CPU 为等待写主存完成所耗费的时间。

至于 Cache 的内容跟不上已变化了的主存内容的问题，一种解决方法是当 IOP 向主存写入（输入）新内容时，由操作系统用某个专用指令清除整个 Cache。这种方法的缺点是使 Cache 对操作系统和系统程序员不透明了。另一种方法是当 IOP 向主存写入新内容时，由专用硬件自动地将 Cache 内对应区域的副本作废，而不必由操作系统干预，从而保持了 Cache 的透明性。前述 CPU、IOP 共享一个 Cache 也是一种办法。

总之，系统结构设计时必须考虑 Cache 的透明性所带来的问题，并予以妥善解决。

3.4.4 任务切换对失效率的影响

由于 Cache 的容量不可能很大（与主存相比），多个进程的工作区很难同时都留驻 Cache。因此，在任务切换时会造成 Cache 失效。失效率大小和任务切换的频度有关，即与任务切换的平均时间间隔 Q_{sw} 有关。

设从 Cache 为空（指新进程所需内容全部不在 Cache 内）到 Cache 全部被装满这一段时间内的失效率为冷启动（Cold-start）失效率；而从 Cache 为现行进程装满后测出的失效率为热启动（Warm-start）失效率。

如果进程切换发生在用户程序因为系统运行管理程序、处理 I/O 中断或时钟中断时， Q_{sw} 值越小，表明由管理程序切换至原来的用户程序越快，Cache 中驻留的原来程序的指令和数据就越多，即失效率越低。如果切换是在几个用户程序之间进行，且每个进程均要更换 Cache 中大部分内容，那么 Q_{sw} 值越小，失效率越高。

Q_{sw} 值一定时，Cache 容量过小，存不下该程序的工作区后，就有很高的热启动失效率。Cache 容量增大会使热启动失效率急剧下降，但增大到基本包含工作区后，热启动失效率的下降渐趋平缓。

在 Cache 容量很小时，由于热启动失效率很高，相对而言，冷启动失效率所占比例很小。因此，增大 Q_{sw} 值（任务切换次数减少）并不能使热启动失效率明显减小，所以总失效率仍很高，且差别不大。当 Cache 容量增大到使热启动失效率迅速下降后，冷启动失效率比重就增大了，此时切换次数起主要作用，增大 Q_{sw} 值会使失效率显著减小。

对于因任务切换而引起的 Cache 失效率可由下述几种方法解决：增大 Cache 容量；改善调度算法，在任务切换时，使有用的信息尽量保存在 Cache 内，不受破坏；设置多个 Cache 用于不同的目的，如设置两个 Cache，一个专用于管理程序，另一个专用于用户程序，在目态和管态之间切换时，不会破坏各自 Cache 中的内容；对于某些操作，如长的向量运算，长的字符串运算等，不经过 Cache 而在主存内直接进行，以避免这些操作由于使用 Cache 而置换出大量有可能重新使用的数据。由于半导体集成电路工艺高度发展，存储芯片单片容量激增，采用增大 Cache 容量以降低失效率，不失为一种行之有效的简便方法。

3.4.5 多处理机系统的Cache结构

多处理机系统一般有两类：一种类型是一个 CPU 和多个 I/O 处理机（通道）组成共享主

存的系统，另一种类型是多个 CPU 组成的系统，或者由多个 CPU 和多个 I/O 处理机组成的系统。这些多处理机系统的 Cache 结构有下述两种：

① 所有处理机共享一个 Cache，它的优点是能保证信息的一致性。然而一个 Cache 的带宽很难满足两个以上 CPU 的访问要求，而且一个 Cache 很难在物理位置上靠近所有的 CPU，从而增大了访问延迟时间。所以，这种结构往往只用于单 CPU、多 I/O 处理机系统。虽然 Cache 在物理位置上与单 CPU 靠近，I/O 处理机访问时延迟会有所增大，但因为是 I/O 数据传送，影响不大。当然，这种结构必须保证 I/O 处理机不会因访问 Cache 失效而影响数据正常传送，这就要求 I/O 系统有足够大的缓冲量，而且还要求 Cache 有足够大的容量和带宽，在出现多个访问 Cache 失效现象时能快速处理。I/O 处理机使用 Cache，会增加 CPU 访问 Cache 的失效效率。不过一般认为，当 I/O 系统访问 Cache 的次数与 CPU 访问 Cache 的次数之比小于 0.05 时，其影响是很小的。

② 每个处理机都有自己的 Cache，如图 3-39 所示。如何保证同一主存单元信息在各个 Cache 的一致性是该结构所要解决的问题。例如，CPU₁ 由主存 m 单元读出一个字（取进 Cache₁），并执行

$$(m)+R_1 \rightarrow m$$

而 CPU₂ 接着执行

$$(m)+R_2 \rightarrow m$$

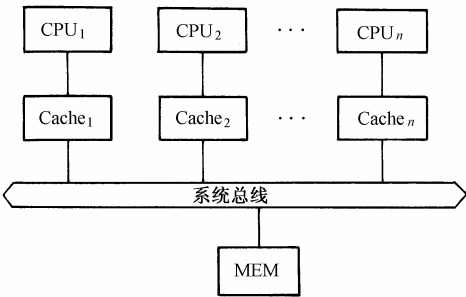


图 3-39 每个 CPU 都有 Cache 的共享主存多处理机系统

若采用写直达法，则主存 m 单元的内容是正确的运算结果，但在 Cache₁ 中与 m 单元对应的单元的内容不是 CPU₂ 运算的最后结果，也即与主存 m 单元内容不一致。解决各 Cache 信息一致性有三种方法：① 播写法。每个 CPU 每次写 Cache 时，把信息播送至所有 Cache 的对应单元，不只是写入自己 Cache 的目标单元。如发现其他 Cache 也有此目标单元，或写入它，或作废它。作废可减少传送的信息量。② 控制某些共享信息（如信号灯或作业队等）不得进入 Cache，从而减少产生不一致性的可能性。③ 目录表法。在 CPU 读、写 Cache 时，若不命中，先查目录表（在主存中），以判定目标单元是否在别的 Cache 内，是否正在被修改等，再决定如何读、写此单元。

3.4.6 “Cache-主存”层次性能分析

评价 Cache，主要看命中率的高低。命中率与块的大小、块的总数（即 Cache 的总容量）、采用组相联时组的大小（组内块数）、替换策略和地址流情况（如簇聚性状况）等有关。

命中率 H_C 与 Cache 的容量、组和块的大小关系如图 3-40 所示。块的大小、组的大小及 Cache 容量这三者增大都会提高命中率。

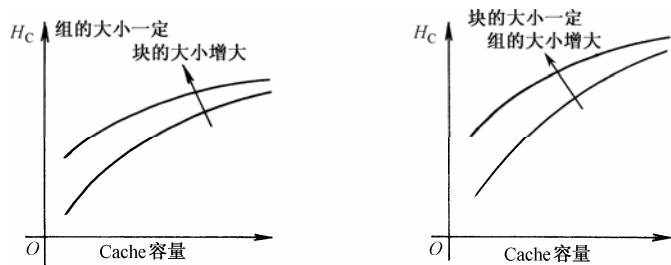


图 3-40 块、组的大小与 Cache 容量对命中率的影响

Cache 在调块时，处理机是空等，所以要求调块尽可能快，从这点看，希望块的大小较小。Cache 容量与不命中率 $(1-H_C)$ 的关系为

$$(1-H_C)=a\times(\text{容量})^b$$

式中， a, b 为常数，且 $b<0$ 。随着存储器芯片集成度提高、价格下降，Cache 的容量不断增大，已达到 128KB，很快就会到几百 KB。

下面分析“Cache-主存”层次的等效速度与命中率的关系。设 t_C 为 Cache 的访问周期， t_M 为主存周期， H_C 为访 Cache 的命中率，则“Cache-主存”层次的等效存储周期

$$t_A=H_C t_C+(1-H_C)t_M$$

因此，系统采用 Cache 后比 CPU 直接访问主存在速度上提高的倍数为

$$\rho=\frac{t_M}{t_A}=\frac{t_M}{H_C t_C+(1-H_C)t_M}=\frac{1}{1-\left(1-\frac{t_C}{t_M}\right)H_C}$$

给定主存和 Cache 速度之比，令 $H_C=\frac{a}{a-1}$ 代入上式，则 ρ_{\max} 与 H_C 的关系如下：

$$\rho=\frac{1}{1-\left(1-\frac{t_C}{t_M}\right)\frac{a}{a+1}}=(a+1)\frac{t_M}{t_M+at_C}$$

因为 $\frac{t_M}{t_M+at_C}<1$ ，因此 $\rho<a+1$

即 ρ 可能的最大值恒比 $a+1$ 小。

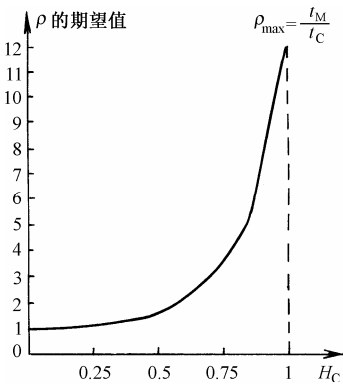


图 3-41 ρ 的期望值与 H_C 的关系

如果 $H_C=0.5$ ，相当于 $a=1$ ，则不论其 Cache 速度有多高， $\rho<2$ ；如果 $H_C=0.75$ ，相当于 $a=3$ ，则 $\rho<4$ ；如果 $H_C=1$ ， $\rho_{\max}=t_M/t_C$ ，这是 ρ 可能的最大值。 ρ 的期望值与 H_C 的关系如图 3-41 所示。由于 Cache 的 H_C 可比 0.9 大得多，能达到 0.996，因此，采用 Cache 结构可使 ρ 接近于所期望的 t_M/t_C 。

H_C 值受 Cache 容量影响很大，当 Cache 容量为 4KB 时， $H_C=0.93$ ；为 8KB 时， $H_C=0.97$ 。若 $t_C/t_M=0.12$ ，则 4KB 的 Cache，其速度提高倍数 $\rho_{4KB}\approx 5.5$ ；8KB 的 Cache，其速度提高倍数 $\rho_{8KB}\approx 6.85$ ，因此，增加 4KB 容量，带来的层次速度提高为

$$\frac{\rho_{8KB} - \rho_{4KB}}{\rho_{4KB}} = \frac{6.85 - 5.5}{5.5} \approx 0.24 (= 24\%)$$

显然，花这个代价而使速度提高 24%是合算的。

对于流水线结构的机器，是否有 Cache、以及 Cache 的容量大小对于机器速度（MIPS）、主存速度均有显著影响。例如，系统时钟为 100 MHz 时（CPU 时钟周期为 10ns），如果没有 Cache，机器速度可能只有 2MIPS，有 4KB Cache，主存周期为 1μs，则机器速度约 5MIPS；同样条件下，Cache 容量增大到 64KB，则机器速度可能达到 15MIPS。

Cache 容量增大，还可以显著降低对主存速度的要求。例如，要达到机器速度为 15MIPS，对于 10ns 的 CPU 时钟周期，4KB 的 Cache，则要求主存访问周期为 200ns；而 Cache 容量增大至 64KB 时，主存周期可降低到 1μs。故不采用 Cache 技术，存储系统速度很难与 CPU 速度相匹配，即很难实现零等待。当 Cache 容量足够大时，主存速度对机器速度的影响会显著减小。

3.4.7 Cache性能计算

存储系统结构性能评价方法主要利用不命中率，因为它是独立于硬件速度的。分立的指令 Cache 和数据 Cache 为各自进行独立的优化提供了可能性。通过分立，可清除因指令和数据冲突而引起的不命中。分立的指令和数据 Cache 同一 Cache 之间合理的比较应该在 Cache 总容量相同的前提下进行。例如，一个分立的 16KB 指令 Cache 和数据 Cache 应该同 32KB 的一体 Cache 相比较。计算分立 Cache 的平均不命中率必须要知道对指令 Cache 和数据 Cache 各自的访问频率。表 3-6 列出了每千条指令不命中率。例如，16KB 指令 Cache 不命中率为 3.82%（即千分之 3.82），16KB 数据 Cache 不命中率为 40.9%，32KB 一体 Cache 不命中率为 43.3%。

表 3-6 千条指令不命中率

容量（KB）	指令 Cache	数据 Cache	一体 Cache
8	8.16	44.0	63.0
16	3.82	40.9	51.0
32	1.36	38.4	43.3
64	0.61	36.9	39.4
128	0.30	35.3	36.2
256	0.02	32.6	32.9

对存储器层次结构评价方法采用平均存储器访问时间：

平均存储器访问时间 = 命中时间+不命中率×不命中代价

不命中代价是指不命中时，存储系统为处理不命中而花费的时间，常用若干时钟周期表示。平均存储器访问时间的单位可以采用绝对时间（如一次命中需时 0.25~1.0ns）或用 CPU 等待存储器的时钟周期数（如不命中代价用 75~100 个时钟周期表示）。平均存储器访问时间仍然是一个间接的性能评价方法，比不命中率好，但不能代替执行时间，该公式可帮助我们 在分立 Cache 和一体 Cache 间做出选择。

【例 3-1】 16KB 指令 Cache 加一个 16KB 数据 Cache 与一个 32KB 一体 Cache 相比较，哪一个具有更低的不命中率？设 36%的存储器访问是数据访问，Cache 命中需要一个时钟周期，不命中代价为 100 个时钟周期。在一体 Cache 中，数据的读出（Load）和写入（Store）需要增加一个时钟周期，因为只有一个 Cache 端满足两个同时发生的请求。请使用表 3-6 中的不命中率计算正确结果。

解：先求得根据表 3-6 转化为不命中率。

$$\text{不命中率} = \frac{\text{表 3.6 不命中率}}{\frac{\text{内存访问次数}}{\text{指令数}}}$$

一条指令只进行一次取指访存操作，故指令不命中率为

$$16\text{KB 指令 Cache 不命中率} = \frac{3.82\%}{1.00} = 0.004$$

由题目可知，36%的指令是数据操作指令，故数据不命中率为

$$16\text{KB 数据 Cache 不命中率} = \frac{40.9\%}{0.36} = 0.114$$

所以，分立 Cache 总不命中率为

$$(74\% \times 0.004) + (26\% \times 0.114) = 0.0324$$

其中， $\frac{100}{100+36} = 74\%$ ，表示访问指令 Cache 的指令占程序总量的比例； $\frac{36}{100+36} = 26\%$ ，表

示访问数据 Cache 的指令占程序总量的比例。

一体 Cache 的不命中率包括指令和数据不命中率，所以

$$\text{一体 Cache 不命中率} = \frac{43.3\%}{1.00 + 0.36} = 0.0318$$

两者相比，一体 Cache（32KB）比分立 Cache 有较低的不命中率。

平均存储器访问时间包括指令访问和数据访问的时间之和：

$$\begin{aligned} \text{平均存储器访问时间} = & \text{指令所占比例} \times (\text{命中时间} + \text{指令不命中率} \times \text{不命中代价}) \\ & + \text{数据所占比例} \times (\text{命中时间} + \text{数据不命中率} \times \text{不命中代价}) \end{aligned}$$

所以

$$\text{分立 Cache 平均存储器访问时间} = 74\% \times (1 + 0.004 \times 100) + 26\% \times (1 + 0.114 \times 100) = 4.24$$

$$\text{一体 Cache 平均存储器访问时间} = 74\% \times (1 + 0.0318 \times 100) + 26\% \times (1 + 0.0318 \times 100) = 4.44$$

由此可见，分立 Cache 的不命中率比一体 Cache 略高，但平均存储器访问时间比一体 Cache 小，是因为分立 Cache 在每个时钟周期提供两个存储器端口，可以避免资源冲突。

由于 Cache 不命中而影响 CPU 性能可用下式描述：

$$\begin{aligned} \text{CPU 时间} = & (\text{CPU 执行程序时间的时钟周期数} + \text{存储器停顿时钟周期数}) \times \text{时钟周期时间} \\ = & \text{指令数} \times \left(\text{每条指令执行的时钟周期数} + \frac{\text{存储器停顿时钟周期数}}{\text{指令数}} \right) \times \text{时钟周期时间} \end{aligned}$$

式中，存储器停顿时钟周期数是由 Cache 不命中带来的。若以不命中计算 CPU 性能，则

$$\begin{aligned} \text{CPU 时间} = & \text{指令数} \times \left(\text{每条指令执行的时钟周期数} + \text{不命中率} \right. \\ & \left. \times \frac{\text{存储器存取次数}}{\text{指令数}} \times \text{不命中代价} \right) \times \text{时钟周期时间} \end{aligned}$$

【例 3-2】 设 Cache 不命中代价为 100 个时钟周期，所有指令都用 1 个时钟周期完成，平均不命中率为 2%，平均每条指令访问存储器 1.5 次，每 1000 条指令平均 Cache 不命中次数为 30。分别用不命中率和每条指令不命中次数计算 Cache 对计算机性能（即 CPU 时间）的影响。

解：用 Cache 不命中时计算的性能为

$$\text{CPU 时间} = \text{指令数} \times \left[1 + \frac{30}{1000} \times 100 \right] \times \text{时钟周期时间} = \text{指令数} \times 4.00 \times \text{时钟周期时间}$$

用不命中率计算的性能为

$$\text{CPU 时间} = \text{指令数} \times (1 + (2\% \times 1.5 \times 100)) \times \text{时钟周期时间} = \text{指令数} \times 4.00 \times \text{时钟周期时间}$$

由于时钟周期时间和指令数是一定的（即无论是否考虑到 Cache 不命中），CPI（每条指令执行的时钟周期数）从 1.00（理想情况下，不考虑 Cache 不命中）增加到 4.00（考虑 Cache 不命中），从而导致 CPU 时间增加 3 倍。如果不采用存储器层次结构，CPI 将会增加到 $1.0 + 100 \times 1.5 = 151$ 倍，这是带有 Cache 系统的近 40 倍。

Cache 不命中对具有较低 CPI 和较快时钟频率的 CPU 有双重影响：CPI 越低，一定的 Cache 不命中时钟数对 CPU 时间的相对影响越大；在 Cache 不命中情况下计算 CPI 时，Cache 不命中代价用不命中时用掉的 CPU 时钟周期数来度量。因此，对于两台有相同存储器层次结构的计算机，时钟频率较高的 CPU 每次不命中所需要的时钟周期数较多，从而使 CPI 花费在存储器部分的周期也较多。

【例 3-3】 设 Cache 为理想状态时，CPI 为 2.0，时钟周期时间为 1.0ns，平均每条指令访存 1.5 次，两个 Cache 容量都是 64KB，块容量为 64B。一个 Cache 采用直接映像，另一个 Cache 采用每组两块的组相联映像。由于组相联必须增加一个多路选择器从某组中选出所需的块，CPU 的时钟周期时间必须扩展 1.25 倍。若 Cache 不命中代价为 75ns，计算两种 Cache 平均存储器访问时间以及 CPU 性能。设直接映像 Cache 不命中率为 1.4%，组相联映像不命中率 1.0%。

解：平均存储器访问时间=命中时间+不命中率×不命中代价

直接映像的 Cache：平均存储器访问时间=1.0+（0.014×75）=2.05ns

组相联映像 Cache：平均存储器访问时间=1.0×1.25+（0.01×75）=2.00ns

$$\text{CPU 时间} = \text{指令数} \times \left(\text{指令执行周期数} + \frac{\text{不命中次数}}{\text{指令数}} \times \text{不命中代价} \right) \times \text{时钟周期时间}$$

$$= \text{指令数} \times \left[\left(\text{指令执行周期数} \times \text{时钟周期时间} \right) + \left(\text{不命中率} \times \frac{\text{存储器访问次数}}{\text{指令数}} \times \text{不命中代价} \times \text{时钟周期时间} \right) \right]$$

式中，根据题意，不命中代价×时钟周期时间均为 75ns。

直接映像 Cache：CPU 时间=I_C×（2×1.0+（1.5×0.014×75））=3.58×I_C

组相联映像 Cache：CPU 时间=I_C×（2×1.0×1.25+（1.5×0.010×75））=3.63×I_C

式中，I_C为指令数，两者比较 $\frac{3.63 \times I_C}{3.58 \times I_C} = 1.01$ 。

解题分析说明，组相联映像 Cache 的存储器访问时间性能较好，而 CPU 性能直接映像 Cache 稍好一些，这是因为组相联映像的指令时钟周期都延长了。由于 CPU 时间是基本评估标准，而且直接映像容易实现，所以本例中直接映像的 Cache 是更好的选择。

上述分析是在 CPU 顺序执行程序前提下进行的，对于乱序执行（如程序转移、调子、中断相应服务等）则如何定义呢？由于指令进入流水线后发生转移必有退出阶段，存储器不命中的全部延迟时间有部分是重叠延迟，所以此时不命中代价定义如下：

$$\text{不命中代价} = \frac{\text{存储器停顿周期数}}{\text{不命中次数}} = \frac{\text{不命中次数}}{\text{指令数}} \times (\text{全部不命中延迟} - \text{重叠不命中延迟})$$

【例 3-4】 沿用例 3-3，设通过加长处理器的时钟周期到 1.25 倍，支持乱序执行。采用直接映像 Cache，不命中代价为 75ns，其中 30%是重叠的，即 CPU 访问存储器平均停顿时间为

$$75 \times (1 - 30\%) = 52.5 \text{ ns}$$

解：乱序执行处理器的平均存储器访问时间 = $1.0 \times 1.25 + (0.014 \times 52.5) = 1.99 \text{ ns}$

CPU 性能（即 CPU 时间）= $I_C \times (2 \times 1.0 \times 1.25 + (1.5 \times 0.014 \times 52.5)) = 3.60 \times I_C$

关于 Cache 性能计算可得到下列一系列公式。

(1) 关于 Cache 索引字段长度计算

$$2^n = \frac{\text{Cache 容量}}{\text{块容量} \times \text{组相联度}}$$

式中， n 是索引字段位数，需取整。组相联度指每组多少块，取 2^i ($i=0, 1, 2, \dots$)。

(2) 关于 CPU 时间计算

CPU 时间 = (CPU 时钟周期数 + 存储器停顿周期数) × 时钟周期时间

存储器停顿周期数 = 不命中次数 × 不命中代价

= 指令数 × 每条指令不命中次数 × 不命中代价

每条指令不命中次数 = 不命中次数 / 指令数

CPU 时间 = 指令数 × (CPI + 平均每条指令的存储器停顿时钟周期数) × 时钟周期时间

CPU 时间 = 指令数 × (CPI + 不命中率 × 平均每条指令访存次数 × 不命中代价) × 时钟周期时间

CPU 时间 = 指令数 × (CPI + 平均每条指令不命中次数 × 不命中代价) × 时钟周期时间

(3) 关于平均存储器访问时间计算

平均存储器访问时间 = 命中时间 + 不命中率 × 不命中代价

此公式不命中代价用时间表示。CPU 时间计算中的不命中代价用时钟周期数表示。

平均每条指令的停顿周期数 = 平均每条指令的不命中次数 × (总的命中延迟

- 重叠的不命中延迟)

(4) 关于多级 Cache 性能计算（以二级 Cache 为例）

平均存储器访问时间 = L1 命中时间 + L1 不命中率 × (L2 命中时间

+ L2 不命中率 × L2 不命中代价)

平均每条指令的存储器停顿周期数 = L1 平均每条指令的不命中次数 × L2 命中时间

+ L2 平均每条指令的不命中次数 × L2 不命中代价

式中，L1 为一级 Cache，L2 为二级 Cache，因为

平均存储器访问时间 = L1 命中时间 + L1 不命中率 × L1 不命中代价

L1 不命中代价 = L2 命中时间 + L2 不命中率 × L2 不命中代价

代入上式，有

平均存储器访问时间 = L1 命中时间 + L1 不命中率 × (L2 命中时间

+ L2 不命中率 × L2 不命中代价)

【例 3-5】 设在 1000 次访存中，一级 Cache 的 L1 有 40 次不命中，在二级 Cache 的 L2 有 20 次不命中。问 L1 和 L2 不命中率分别为多少？设 L2 到主存的不命中代价为 100 个时钟周期，L2 命中时间为 10 个时钟周期；L1 命中时间是一个时钟周期，每条指令的存储器访问次数为 1.5，求平均存储器访问时间和平均每条指令的存储器停顿周期数（写操作影响不计）。

解：L1 不命中率为 $\frac{40}{1000} = 4\%$ 。因为 L2 有 20 次不命中，对于“L1-L2”两级 Cache 的

L2 局部不命中率为 $\frac{20}{40} = 50\%$ ，L2 的全局不命中率为 $\frac{20}{1000} = 2\%$ 。

平均存储器访问时间=L1命中时间+L1不命中率×(L2命中时间+L2不命中率×L2不命中代价)
 $=1+4\% \times (10+50\% \times 100) = 3.4$ 个时钟周期

因为每条指令访存次数为 1.5, 1000 次访存, 相当于执行了 $\frac{1000}{1.5} = 667$ 条指令。667 条指令

令 L1 的不命中次数为 40 次, 则 1000 条指令折算的不命中次数为 $\frac{1000}{667} \times 40 = 60$ 次。同理, L2 不命中次数为 $\frac{1000}{667} \times 20 = 30$ 次。设数据和指令的不命中次数相等, 则每条指令的平均存储器停顿周期数为

平均每条指令的存储器停顿周期数=L1 平均每条指令的不命中次数×L2 命中时间
 + L2 平均每条指令的不命中次数×L2 不命中代价
 $= \frac{60}{1000} \times 10 + \frac{30}{1000} \times 100 = 3.6$ 个时钟周期

如果从平均存储器访问时间减去 L1 命中时间, 再乘上每条指令的平均访存次数, 也能得到相同的平均每条指令的存储器停顿周期数 $(3.4-1.0) \times 1.5 = 2.4 \times 1.5 = 3.6$ 个时钟周期。

由此可见, 在多级 Cache 中, 用每条指令不命中次数进行计算要比用不命中率进行计算来得更清楚。

上述公式中描述的访存操作是读操作和写操作的结合, 并假定 L1 采用写回法。当采用写直达法时, L1 将会把所有写操作发送给 L2, 不仅仅指不命中时, 而且要用到写缓冲区。经过对两级 Cache 的容量和不命中率关系以及 L2 容量对相对执行时间关系的统计分析, 可得出如下结论:

(1) 当 L2 容量比 L1 容量大许多时, 全局 Cache 不命中率接近 L2 不命中率, 因此, 对于单级 Cache 的分析也适用。

(2) L2 的局部不命中率是 L1 不命中率的函数, 它会随 L1 的变化而变化, 这不是一个好的度量方法。因此, 对两级 Cache 评估时应该用全局 Cache 不命中率。

【例 3-6】 设 L2 直接映像时命中时间为 10 个时钟周期, 当采用 2 路组相联时命中时间为 10.1 个时钟周期。L2 直接映像的局部不命中率为 25%, 2 路组相联时的局部不命中率为 20%, L2 的不命中代价为 100 个时钟周期, 分析 L1 的不命中代价。

解: L2 采用直接映像, 则 L1 不命中代价为 $10+25\% \times 100 = 35$ 个时钟周期。

L2 采用 2 路组相联, 则 L1 不命中代价为 $10.1+20\% \times 100 = 30.1$ 个时钟周期。

由于 L2 总是和 L1 以及 CPU 同步, 因此, L2 命中时间必须是时钟周期的整数倍。以 L2 命中周期为 10 个周期或 11 个周期计算, 则 L2 不命中代价分别为

$$10+20\% \times 100 = 30 \text{ 个时钟周期}$$

$$11+20\% \times 100 = 31 \text{ 个时钟周期}$$

由此可见, 通过减小 L2 的不命中率, 可以降低不命中代价。

目前, 可采用下列几种 Cache 优化策略:

(1) 减少不命中代价。可采用多级 Cache、关键字优先、读不命中优先、合并写缓冲区、牺牲 Cache (Victim Cache) 等方法。牺牲 Cache 存放被替换出去的块, 下次不命中时, 先检查牺牲 Cache, 如果找到, 该牺牲块与 Cache 块对调。

(2) 降低不命中率。可采用增大块容量、增大 Cache 容量、增加相联度、路径预测和伪相联、编译优化等方法。

（3）通过并行技术降低不命中代价和不命中率。可采用无阻塞 Cache、硬件预取、编译预取等技术。

（4）减少 Cache 命中时间。可采用小而简单的 Cache（避免地址转换）、流水线式 Cache、跟踪 Cache 等方法。跟踪 Cache 内有 CPU 执行指令的动态轨迹，可预测动态指令序到。

表 3-7 反映了几种优化 Cache 技术对 Cache 复杂度和性能的影响。表中的“缺失代价”即“不命中代价”，“缺失率”即“不命中率”，“+”表示增加，“-”表示减少，硬件复杂度中 0 表示最简单，3 表示极难。

表 3-7 Cache 优化技术总结及对 Cache 复杂度和性能的影响

技 术	缺失代价	缺失率	命中时间	硬件复杂度	备 注
多级 Cache	+			2	硬件开销大,当 L1 容量与 L2 不等时实现较难, 广泛使用
关键字优先与提前重启动	+			2	广泛使用
给出读缺失相对于写的优先级	+			1	对单处理器而言效果较小, 广泛使用
合并写缓冲	+			1	在 21164、UltraSPARCIII 中与写直达一起使用, 广泛使用
牺牲 Cache	+	+		2	AMD Athlon 中使用, 容量大小为 8 条目
增大块容量	-	+		0	作用较小, Pentium4 L2 块容量为 128B
增大 Cache 容量		+	-	1	广泛使用, 尤其在 L2 中
增大相联度		+	-	1	广泛使用
路预测 Cache		+		2	在 UltraSPARCIII 一级 Cache, MIPS R4300 系列的 D-Cache 中使用
伪相联		+		2	在 MIPS R10000 的一级 Cache 中使用
采用编译技术		+		0	软件设计是关键, 一些计算机提供编译器选项
非阻塞 Cache	+			3	乱序执行 CPU 中都使用
硬件预取	+	+		对指令为 2 对数据为 3	一般预取指令, UltraSPARCIII 中预取数据
编译控制的预取	+	+		3	需要非阻塞 Cache, 有一些处理器支持
小而简单的 Cache		-	+	0	效果较小, 广泛使用
Cache 索引中避免地址转换			+	2	Cache 容量小时效果较小, 在 Alpha21164 UltraSPARCIII 中使用
流水线 Cache 访问			+	1	广泛使用
跟踪者 Cache			+	3	在 Pentium 中使用

3.5 虚拟存储器

3.5.1 虚拟存储器基本结构和工作原理

虚拟存储器是指“主存-辅存”层次，它能使该层次具有辅存容量、接近主存的等效速度

和辅存的每位成本，它使程序员可以按比主存大得多的虚拟存储空间编写程序（即按虚存空间编址）。只要主存容量大于某个最小值，不论机器配备多大容量的主存，程序可不作任何修改照样运行。当然，主存容量会影响系统工作效率，如果程序过大而主存容量过小，则运算速度会明显下降。虚拟存储器与 Cache 有许多相似之处，但也有重要区别。

首先，Cache 的主要作用是弥补主存和 CPU 之间的速度差距，因此它的管理部件是用硬件实现的，并对程序员透明。但虚拟存储器的主要作用是弥补主存和辅存之间的容量差距，因此它的管理部件基本上靠软件，适当结合硬件来实现，并且虚拟地址空间可被应用程序员感觉到和加以利用，而它的实现对系统程序员也不是透明的。

由上述不同目标出发，系统设计条件也不同。Cache 访问时间与主存访问时间相比通常在 1:5~1:10 之间，每次传送的信息单位（块）也比较小，只是几至几十字节。而虚拟存储器访问时间比（指“辅存-主存”相比）要大得多，达到 100:1~1000:1 以上，每次传送的信息单位（字段或页面）也很大，有几十至几千字节。

Cache 为了争取速度，CPU 对 Cache 和主存均建立了直接访问路径，但虚拟存储器不一样，辅存必须经过主存才能和 CPU 通信。

在原理上，虚拟存储器和 Cache 有许多是相同的。例如，它们都把信息分成基本单位——块或页，作为一个整体从慢速存储器调入快速存储器，供 CPU 使用。它们都要遵循一定的映像函数安排信息块在快速存储器内的位置。当快速存储器已占满，且 CPU 访问不命中时，则需依据确定的替换策略，成块更新快速存储器中内容。为了获得较高命中率，它们都要利用程序访问局部化性质，寻求上述问题的最佳解决方案。

地址映像与变换、替换算法及其实现等基本概念，在前面已做过讲述，本节着重讲述虚拟存储器基本原理和工作过程，并对各种情况、各个工作阶段的速度要求和实现方法进行分析。

3.5.2 虚地址和辅存实地址的变换

对具有虚拟存储器的计算机系统来说，程序员是按虚存空间编制程序，即按机器指令地址码编制，该地址码就是虚地址，由虚页号和页内地址组成，如图 3-42 所示。

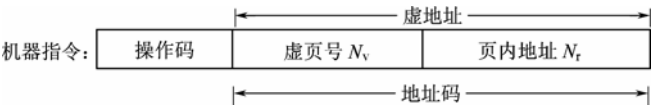


图 3-42 虚地址组成

虚地址是辅存的逻辑地址，而不是辅存的实地址。以磁盘为例，按字节编址的实地址 N_d 如图 3-43 所示。

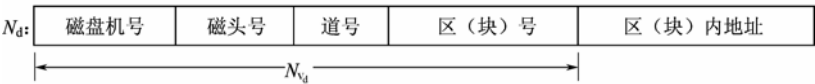


图 3-43 实地址示例

辅存逻辑地址和辅存实地址两者不同，因此在虚拟存储器中有虚存空间到辅存实空间的地址变换。辅存一般是按信息块编址，而不是按字节编址，若使一个块（区）的大小等于一个虚页面的大小，则只需由虚页号 N_v 变换到 N_{v_d} 即可完成虚地址到辅存实地址的变换。为此，

可以采用前述页表的方式。由 N_v 变换到 N_{v_d} 的表称为外页表，由 N_v 变换到主存实页号 n_v 的表称为内页表（即前述页表）。

外页表的容量与内页表一样，也是 2^{N_v} 。可采用前述表层次技术。前面讲过每访问一次主存都要将虚地址变换到主存实地址，只有当出现页面失效时（其概率不到 1%）才需调用外页表进行虚地址到辅存实地址的变换，以便由辅存将该虚页调入主存。因为访问辅存需经机械动作，速度低，费时多，因此查外页表的速度也较低，外页表可存于辅存，只在需要时才调入主存。在页面失效时，处理机不空等该页由辅存调入主存，而是通过程序换道，切换到另一个已在主存的用户程序。虽然换道需要执行相当多的指令，但比起调页时辅存机械动作花费的时间还是少得多。

虚地址到辅存实地址的变换过程如图 3-44 所示。其中，装入位表示该信息块（区）是否已由海量存储器（如磁带）装入磁盘。装入位为 1，表示外页表内的 N_{v_d} 为有效辅存（磁盘）的实地址。虚地址到辅存实地址的映像采用“全相联”方式，其变换用软件实现。

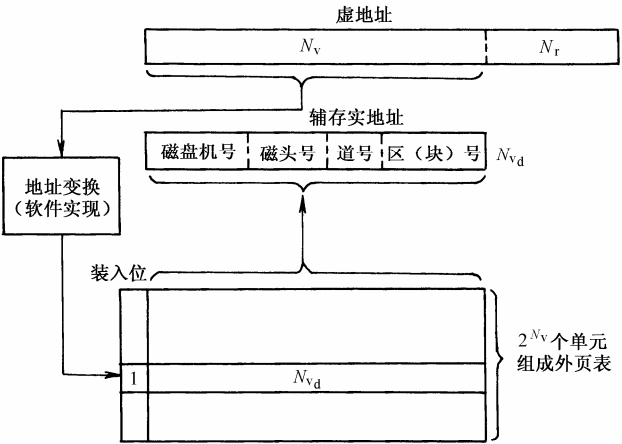


图 3-44 软件查外页表由虚地址变换到辅存实地址

3.5.3 多用户虚拟存储器

上述是按多用户共用一个总的虚存空间来分析的，虚存空间总共有 2^{N_v} 页，由多个用户（多道）分用这个虚存空间。由于每个用户（道）不具有与其他用户（其他道）完全无关的编址空间，这就给编制程序带来了麻烦。所以，后来的虚拟存储器都设计成每个用户具有独立的有 2^{N_v} 个页面的虚存空间，并增加用户标志位 u （其宽度为 u 位）来对应 2^u 个用户，用以区分各个用户在辅存所占的空间。辅存逻辑地址（即多用户虚地址） N_s 由 u 和指令地址码 N 两部分组成，如图 3-45 所示。

多用户虚存总空间为 2^u 个用户空间，每个用户空间为 2^N 个单元，共有 $2^u \times 2^N$ 个单元。多用户虚地址与主存实地址的差 $n_d = N_s - n_p$ ，虚页号 $N_v = N_s - N_r$ ，而单用户中的 $N_v = N - N_r$ ，现用 $N_v' = N - N_r$ 表示。由虚地址变换成主存实地址就是将 $N_v = u + N_v'$ 变换成 n_v 再拼接上 N_r 。

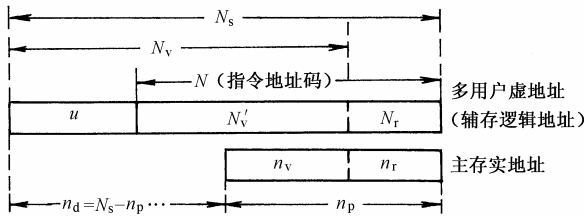


图 3-45 多用户虚地址的组成

如果采用“全相联页式管理”查表法，每个用户（每道程序）有其内页表（由 2^{n_v} 个单元组成），且已在主存，则其地址变换如图 3-46 所示。由 u 指明该用户内页表的起始地址，由 N'_v 提供内页表内对应单元（行）的偏移量，取出该单元内容，判断其装入位是否有效。若有效，则取其实页号 n_v ，再与 N_r 拼接成主存地址；若无效，则进入页面失效处理。

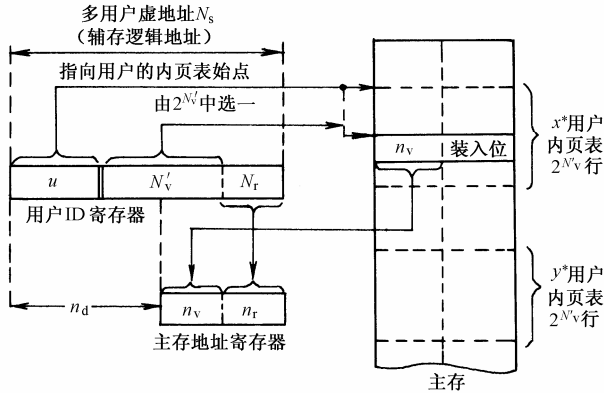


图 3-46 多用户全相联页式管理中将 N_s 变换成主存地址

对于“全相联段页式管理”， N 是多用户虚地址的指令地址码， N'_v 分成段号和页号，采用段表和页表层次， u 指 x 用户的段表的起始地址，段号 y 为该段表内偏移量，取出内容为 x 用户 y 段的页表的起始地址，页号为该页表内偏移量，取出该单元内容，即为实页号 n_v ，再与 N_r 拼接成主存地址，如图 3-47 所示。注意，对多用户虚拟存储器，每个用户的编址空间和总的虚存空间（即辅存空间）是不同的。

虚拟存储器各部分综合成图 3-48。虚拟存储器工作全过程如下：

- (1) CPU 取指，得到用户虚地址 N_s （用户标志 u +用户虚页号 N'_v ），送内部地址变换。
- (2) 在内部地址变换中，依据 $u+N'_v$ 查内页表，取出内页表对应单元（行）内容，判断其装入位是否为 1，若是，则进入（3）；为 0，则进入（4）。
- (3) 内页装入位为 1，说明页面有效，则将内页表对应单元（行）内容 n_v 与虚地址 N_s 中的 N_r （即 n_r ）拼接成主存实地址 n_p ，然后在主存内完成读/写操作。
- (4) 内页装入位为 0，说明该页未在主存中，产生页面失效中断，然后转入（5），启动外部地址变换，同时转入（9），启动查实存页表。
- (5) 依据“用户标志 u +用户虚页号 N'_v ”（即二者拼接）在外部地址变换中形成辅存实地址。

- (6) 判断该页是否在辅存，若在辅存，则将辅存实地址送辅存硬件；若不在辅存，则引发中断，转入(8)。
- (7) 在 I/O 处理机(通道)参与下，将该页从辅存调入主存。该操作与 CPU 的运行并行进行。
- (8) 该页不在辅存，则转入中断，从海量存储器(如磁带)将该页调入辅存。
- (9) 页面失效时，进入查实存页表(LRU 页表)，该操作与(5)，(6)并行。
- (10) 查实存页表得到主存未满结果，则将内部地址变换后得到的实页号送实存页表(即写入 LRU 页表)。
- (11) 查实存表得到主存已满结果，则执行 LRU 算法。
- (12) 找到被替换的页，得到该页的实页号。

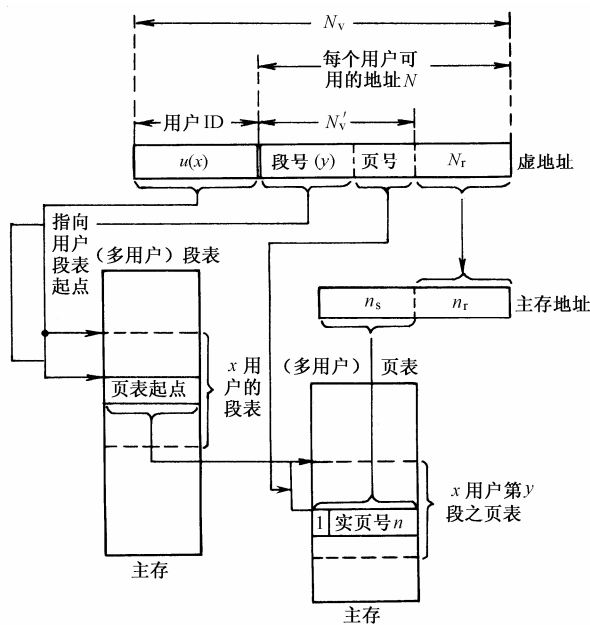


图 3-47 多用户段页式管理中将 N_s 变换成主存地址

- (13) 将实页号(主存未满时为内部地址变换后得到的实页号;主存已满时为执行 LRU 算法后得到的被替换的实页号)送 I/O 处理机。I/O 处理机根据辅存实地址到辅存读出一页信息，然后根据主存实页号(实地址)将该页信息写入主存。
 - (14) 在页面替换时，如果被替换的页调入主存后一直未经修改则不需回送辅存；如果已经修改(查主存页面表可知)，则需先将其送回辅存原来位置，而后再把调入页装入主存。
- 页面失效处理是设计存储体系的关键之一，考虑不周会使存储体系无法正常工作。前面已叙述，页面划分只是机械地对虚、实存空间进行等分，与程序的逻辑结构毫无联系。由于存储器是按字节编址的，因此可能出现指令、操作数、字符串跨页存储的情况。还有，对于间接寻址方式，尤其是多重间接寻址方式，在间接寻址过程中，可能出现跨页(甚至跨多页)访问，故页面失效中断完全可能出现在取指令、取操作数、间接寻址等过程中，即页面失效中断会在一条指令的分析、执行过程中发出，而且必须立即响应和处理，否则该条指令无法执行下去，这就违反了通常应在一条指令执行结束才响应中断的惯例(事实上，它是由 CPU

硬件逻辑结构和时序所决定的，总线请求例外)。由此就引起了在处理主存页面失效后，如何恢复中断现场、回到断点的问题，这需要软、硬结合进行。处理该问题一般有下列三种方法：

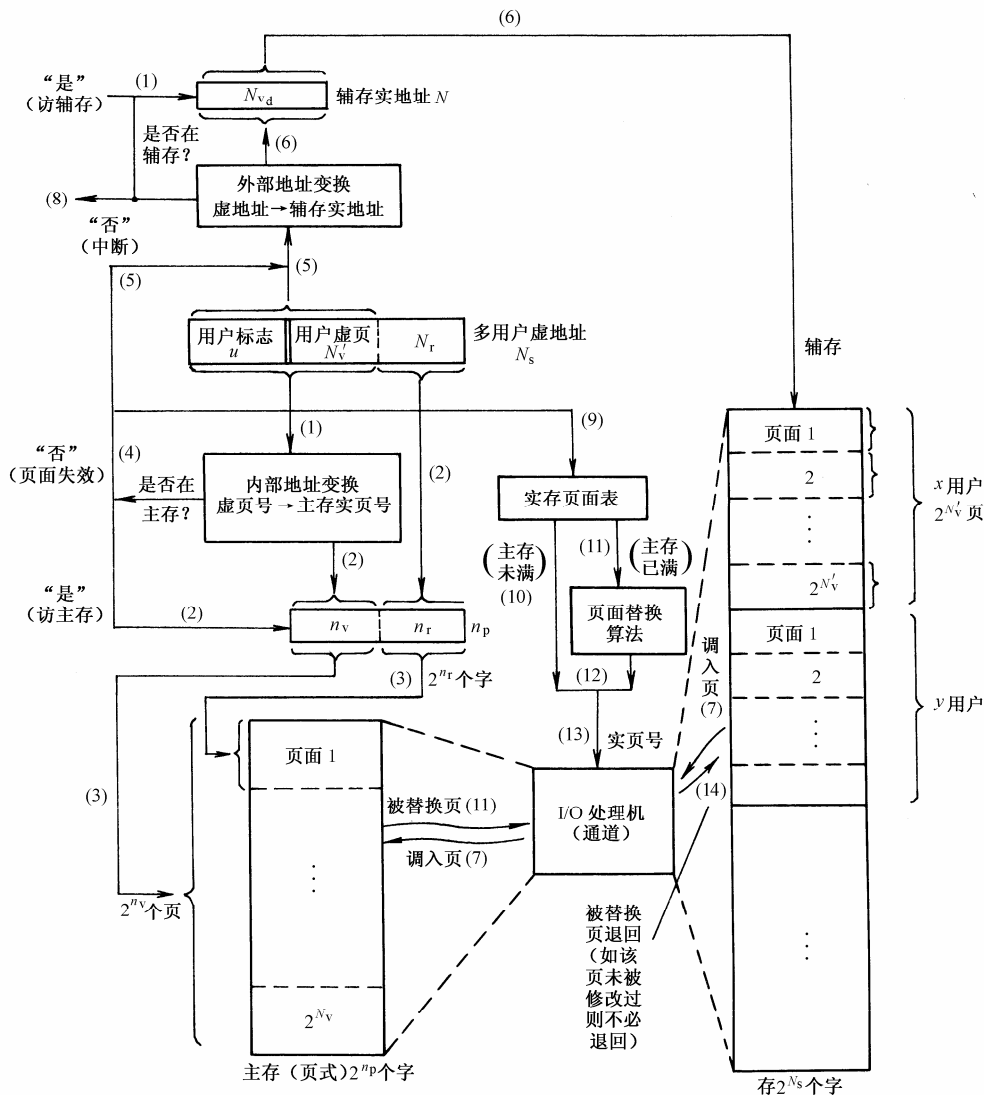


图 3-48 多用户虚拟存储器工作全过程

(1) 出现页面失效中断时，用后援寄存器技术把该条指令的现场全部保存下来，处理完该中断，且把所需页调入中断后，能由该指令从出现页面失效点处继续执行该指令。

(2) 在页面失效中断的现场保存下来后，进行调页操作，并从头再执行该条指令。

(3) 在执行字符串指令前，预判字符串操作数的首、尾字符所在页是否已在主存，只要有一个还没有装入主存，则发出页面失效中断，待该页调入后，才开始执行这条字符串指令。这种方法只适用于字符串长度不超过一页的情况。

上述跨页处理中需注意替换算法的设计，不能出现“颠簸”，即新调入的页不能把原已在主存内、且当前要使用的页替换出去。若遇到指令和两个操作数都是跨页存储时（这是最

坏情况，其概率很低），这条指令执行就要用到 6 个实页。因此，分配给各道程序的实页数要考虑上述情况。

3.5.4 加快地址变换的方法

由上述虚拟存储器工作全过程可知，它有两种地址变换：一种是由虚地址变换成主存实地址，每访主存一次就要进行一次，故这种内部地址变换要求变换速度很高；另一种是只有当产生页面失效时才需进行的虚地址到辅存实地址的变换，这种外部地址变换速度可低些。至于替换算法的实现，由于它仅出现于页面失效且主存已满时，故其概率比外部地址替换还要低，所以对它的速度要求可更低一些。就虚拟存储器的速度要求而言，关键在于虚地址到主存实地址的变换速度，这个速度如果达不到要求，则虚拟存储器无法使用。

内部地址变换是通过查表得到实地址的，不同的表结构，其查表速度是不一样的。对于页表法，由于页表存于主存中，因此每访主存一次，为查页表就要再增加一次访存。如果采用段页式，仅查表就需访存二次。所以为存、取一个单元，就需访存三次，比不采用虚拟存储多了二次，从而使速度严重下降。对于相联目录表法，在全相联时需要对 2^{n_v} 行相联查找，也很费时间。

在实际查表过程中，由于程序局部性的特点，对表内各行的使用不是随机性的，而是聚簇性的，即不论哪种表结构，在一段时间内，实际上只用到表内很少几行。因此，不采用有 2^{n_v} 行的全“相联目录表”，而是用快速硬件构成比全表小得多的部分“相联目录表”（如只有 8~16 行），这个部分目录表称为快表，而全表称为慢表。快表查找速度比慢表快得多，只是当快表查不到时，才从容量为 2^{n_v} 行的慢表（内页表）中找出对应的实页号。慢表存在主存内。运用快表和慢表实现内部地址变换如图 3-49 所示。

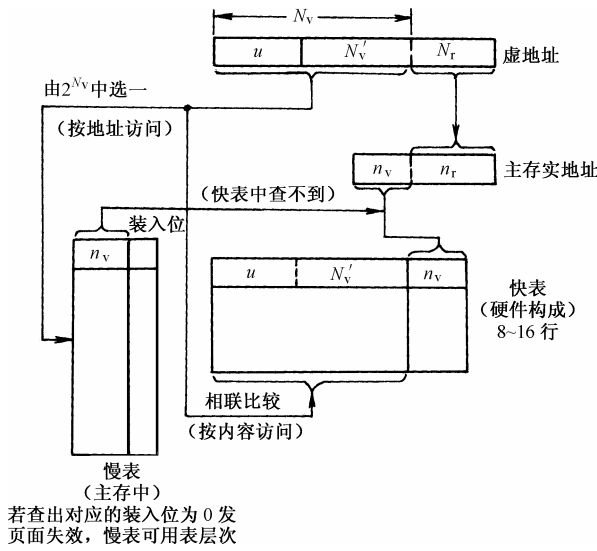


图 3-49 用快表和慢表实现内部地址变换

查表时，由虚页号 ($N_v=u+N_v'$) 同时去查快表和慢表。如果在快表中查到，则能很快地得到实页号 n_v 并送主存实地址寄存器，且使慢表的查找作废，从而达到虽使用虚拟存储器但访主存速度几乎没有下降的目的。如果快表查不到，就费一个访主存周期查慢表，查到的实页号

n_v 送主存实地址寄存器，同时将此虚页号和对应的实页号送入快表，替换快表中的内容。

虚拟存储器只是在有了快表以后才得以真正实用。快表和慢表实际上构成了层次，与“主存-辅存”层次在概念上是相同的，因而前述替换算法也适用于它，一般采用 LRU 法。如果是堆栈型的 LRU，则快表容量越大，其命中率越高；但容量越大，其相联查找速度也越慢，所以快表的命中率和查表速度是矛盾的。若快表取（8~16）行，每页容量为（1~4）KB，则快表容量可反映主存中的（8~64）KB，一般来说，这样的快表，其命中率是不会低的。为了解决快表的容量（命中率）与速度之间的矛盾，可用速度比相联存储器快得多的按地址访问存储器芯片构成容量更大的快表，并采用 3.1.7 节所述的用硬件实现的散列法，使快表地址从 N_v 压缩到 $A=H(N_v)$ 。然而，散列不是唯一的，即 A 与 N_v 不是唯一对应的，这就存在可能查错的问题。为此，在快表内加上 N_v 项。在用散列法求得 A 并查到 n_v 后，将快表内同一行的 N_v 与虚地址中的 N_v 相比较。若不等，就表明出现了散列冲突，即 A 地址单元内的 n_v 不是虚地址对应的实页号，这时就靠查慢表获得 n_v 。判相等可与访问主存并行进行。这样构成的快表其容量比相联查找法大，可达 64~128 行，从而可进一步提高快表命中率，并且仍有很高的查表速度，其原理如图 3-50 所示。

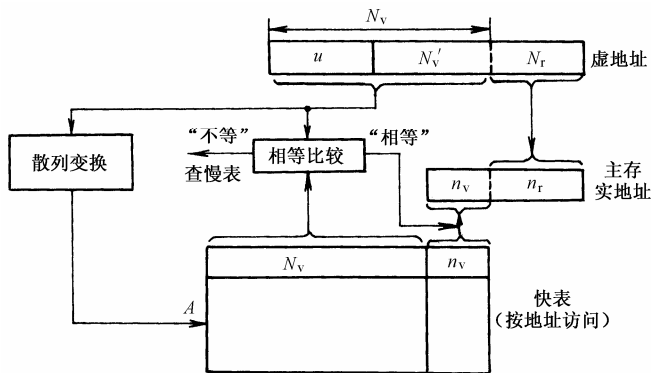


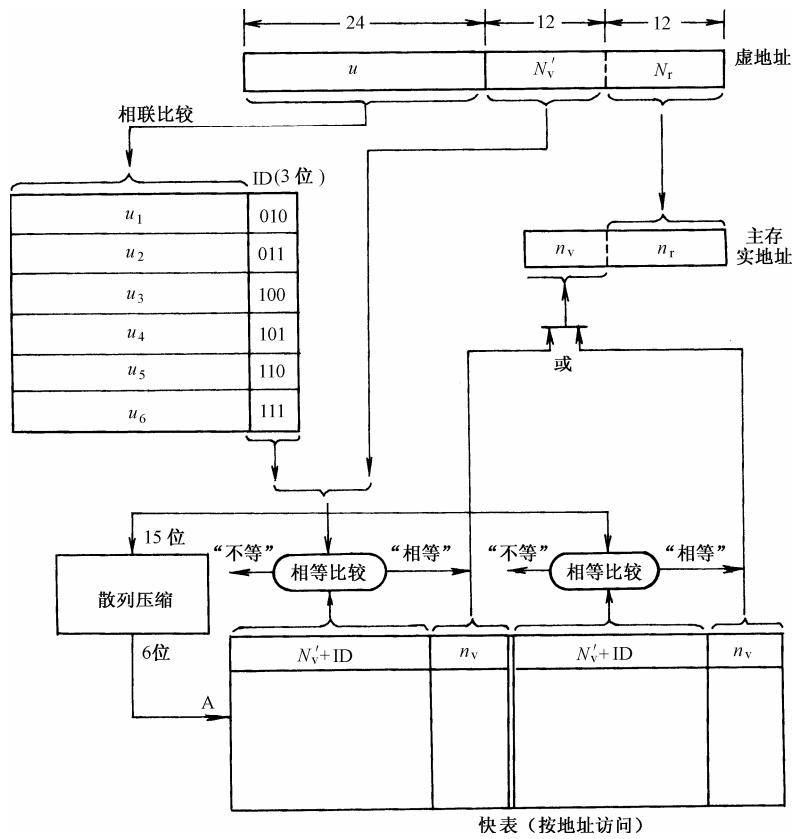
图 3-50 用散列法实现快表按地址查找

上述方法的出发点是让概率高的操作尽可能用最快的方法处理，即使不正确或出错，只要出错概率很低且能发现和纠正，即使这种纠正要花费较多时间，也仍然会带来系统整体效率的提高。这是哈夫曼（Huffman）概念的一种应用，也是当代机器设计的重要思路。

IBM 370/168 的虚拟存储器快表结构是一个典型例子。它在采用散列法的基础上，增加两项措施，见图 3-51。一项措施是使每个地址 A 对应两个虚页号，即把虚页号 $1(N_v' + ID)_1$ 和虚页号 $2(N_v' + ID)_2$ 放在同一个 A 单元内，用两套相等比较电路，哪一个相符就送哪一个 N_v ，只有当 A 单元的两个虚页号都不相符时才不命中，需要查慢表来取得 N_v 。另一项措施是把虚地址中的用户标志 u 进行压缩。因为散列变换（压缩）的入、出位数差越小，散列冲突概率就越低。而在 IBM 370/168 中 u 长达 24 位，对应于 2^{24} 个任务（用户），但在比较长的一段时期内仅几个任务在运行，远比 2^{24} 小得多，即在同一时期内 u 的变化概率比起 N_v' 的变化概率要小得多，故只需把当前正在运行的那几个用户标志 u 值存在高速相联寄存器内（如 6 个），用三位 ID 表示，从而实现了将 24 位的 u 压缩成三位 ID。在进行地址变换时，先将虚地址中的 u 在相联寄存器组内进行查找，找到相应的 ID，再与 N_v' 拼接，由原

来的 $u + N_v' = 36$ 位压缩成 $ID + N_v' = 15$ 位，从而，既能缩短相联比较位数，缩小散列变换的入、出位数差，又能在任务切换时，不易出错。这样，在快表内可以同时存在多达 6 个任务，在任务切换时，不必由操作系统使整个快表或某行内容作废，真正实现快表对操作系统和系统程序员是“透明”的。只有当 u 值与 6 行相联寄存器组的任一个都不相符，即表明已切换到 6 个之外的新任务时，才需用替换算法将新任务的页表内容取入快表。所以快表内容是随着任务切换逐行地自动更换，而且常用任务（或主任务）被替换出去的概率很低，从而解决了因快表内容全部作废所带来的虚、实地址变换速度下降的问题。

在 IBM 370/168 之后的机器，快表结构基本上与 IBM 370/168 相似。该例子说明了把软件概念（如散列技术）应用于系统设计所带来的好处，以及软硬相互渗透，设计出好的“透明”硬件对提高计算机系统性能的意义。



如图 3-51 IBM 370/168 虚拟存储器快表结构

3.5.5 虚拟存储器性能分析

1. 主存空间利用率

主存容量有限（与辅存、海量存储器相比而言），减少主存空间的浪费，提高其利用率是设计存储体系的指标之一。主存空间利用率是指用户程序占据的主存空间与分配给该用户的主存容量之比。

设 S_s 为用户程序的平均长度（按字节编址），页面长度为 S_p ，当 $S_s \gg S_p$ 时，页内零头的

平均值为 $S_p/2$ 。所以 S_p 越大零头损失越大。从这一点出发，页面小可以提高主存利用率。然而，程序需要有页表（内、外页表），其大小为 S_S/S_p ，若页面过小，就需用很大的页表，这会降低主存利用率。因此，页面过大或过小，都会降低主存空间利用率。在主存内非程序占据的空间（页内零头和页表）共为

$$S_u = \frac{S_p}{2} + \frac{S_S}{S_p}$$

则对应该程序的主存空间利用率为

$$\eta = \frac{S_S}{S_S + S_u} = \frac{2S_S S_p}{S_p^2 + 2S_S(1 + S_p)}$$

η 最大值应对应于 S_u 的最小值，而对应最小 S_u 的 S_p 就是最佳页面大小 S_p^{OPT} ，所以对 S_p 求导有

$$\frac{dS_u}{dS_p} = \frac{1}{2} - \frac{S_S}{S_p^2} = 0$$

得

$$S_p^2 = 2S_S$$

所以

$$S_p^{OPT} = \sqrt{2S_S}$$

代入 η 式，化简后得

$$\eta^{OPT} = \frac{1}{1 + \sqrt{2/S_S}}$$

对应不同的 S_p 值的 η 与 S_S 的关系曲线，见图 3-52。如果只从主存空间利用率考虑，页面大小应按 $\sqrt{2S_S}$ 选取，但这个值往往比实际所用 S_p 值要小，这是因为实用 S_p 大小的确定需考虑多种因素。实用 S_p 大小的确定方法如下：首先从用户程序占用的 S_S 出发，计算 S_p^{OPT} ， η^{OPT} ，然后，考虑其他因素，对 S_p^{OPT} 予以调整。例如，磁盘查找时间要比传送时间长得多，不论页面多大，其查找时间都一样，为了提高调页效率，应使 S_p 增大。又如，可从降低指令、操作数和字符串跨页存储的概率来调整 S_p ，也可以由操作系统为页面替换而花费的时间来调整 S_p 。此外， S_p 还应联系提高主存命中率的需要来确定。总而言之，一个合适的、实用的 S_p 是各方因素综合平衡的结果。

2. 主存命中率

主存命中率 H 是评价存储体系的重要指标，而页面大小 S_p 和分配给该道程序的主存容量 S_1 ($S_1 = S_S = S_u$) 是影响命中率 H 的重要因素，其典型关系如图 3-53 所示。

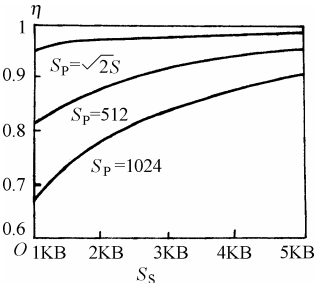


图 3-52 η , S_p , S_S 关系曲线

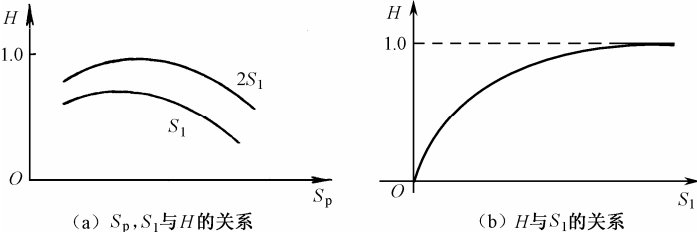


图 3-53 S_p , S_1 , H 关系图

对给定的 S_1 ，随着 S_p 的增大， H 先增大，达到某个最大值后，随 S_p 增大反而减小。其原因如下：设任务地址流 A 中相邻逻辑地址之间的距离为 d_r ，在 S_p 较小时，若 $d_r < S_p$ ，则随

着 S_p 增大，同页内访问的概率会提高，从而 H 随着 S_p 增大而上升。若 $d_r > S_p$ ，两个地址相隔较远，不在同一页，如果另一页也在主存内，那当然会命中。然而，对于堆栈型 LRU 替换算法，在分配给该程序的主存容量 S_1 一定时，增大 S_p ，则总页数要减小，这会使命中率 H 下降（见 3.2 节所述）。所以，当 S_p 较小时，前一种因素起主要作用， H 随着 S_p 增大而上升；当 S_p 达到某值之后，后一种因素起主要作用，页数明显减小，导致增大 S_p 反而使 H 下降。当然，从图上可见，增大分配给该道程序的容量 S_1 能延缓后一种因素所引起的 H 下降。

S_1 与 H 的关系如图 3-53 所示。增大 S_1 在开始时作用明显，当 S_1 值取得能把整道程序全部装入时， H 达到最高值。然而，同一时期只有一部分主存在用，必然使主存利用率 η 变坏，这也不符合采用存储层次的目的。因此， S_p 、 S_1 的选择都是折中平衡，目标是使系统的性能价格比尽可能高。目前还没有确能反映实际状况的分析模型，以致不能定量地确定 S_p 的值。就虚拟存储器的命中率 H 和主存利用率 η 来看， H 是更重要的。所以 S_p 应以 H 为主选定，但要兼顾到 η ，不能使 η 太低。目前，一般机器的 S_p 取 1KB~4KB。除了 S_p 、 S_1 对 H 有影响，替换算法也对 H 有重要影响，这在讲述替换算法的 3.2 节已经分析过了。

以上讨论虽然联系了多道程序的情况，但主要是围绕单道运行的情况来分析。从多道运行角度分析，影响的因素更多。

对分时系统，分配给 CPU 的时间片大小会影响虚拟存储器的使用。时间片较小，应尽量减少页面交换次数，此时， S_p 应较大，否则时间片的大部分会被调页消耗掉。对 S_1 却可降低，因为时间片的限制，来得及使用的主存容量会较小。当然 S_p 也不能过大，不然会出现连一页也没有用完就得换道的情况。

对多任务系统，因为每道占有的主存页面数比只运行单道程序时要少，这会使命中率 H 下降。但在调页时，CPU 可切换到另一道程序，不是空等，而是与 I/O 处理机并行工作，CPU 的效率不会降低。因此，多道时 CPU 效率比单道时高。但作业数 J 不能无限增加，否则适得其反。用概率模型进行模拟分析，当 $J < J_{OPT}$ 时，因为调页和 CPU 并行工作而使 CPU 效率随 J 增大而增加；但当 $J > J_{OPT}$ 后， J 的增加会使每个作业的主存分配量过小而使 H 过分下降，从而使 CPU 效率下降。

为了在多任务系统中优化 CPU 效率，人们提出了各种调页模型。第一种认为，如果调页操作能使辅存约有一半时间是忙着的，则 CPU 效率为最大（即 50% 准则）。这里多道程序的道数和随之而来的为每道程序的主存分配量明显影响页面失效率。第二种认为，调页时间近似等于二次页面失效之间的平均时间时，CPU 效率最高。第三种认为，每道程序的页面分配量应选择能使二次页面失效之间的平均时间达到最大值。按照这些观点可提出调整道数（并行作业数）的算法，以及使系统吞吐量最大的存储管理策略。

上述都是按虚存空间大于实存空间来讲述的，这是虚拟存储系统的基本特征。但也有一些小型机是把小的虚存空间映像到大的实存空间，这能使机器的实际主存空间大于 CPU 直接扫描的存储空间，是扩大机器存储功能的一种办法，而且便于实现快速的虚、实地址变换（因为 $N_v < n_v$ ，页表入口 2^{N_v} 相对较小，便于用高速随机访问存储芯片构成）。

3. “Cache-主存-辅存”层次

目前计算机系统的存储体系几乎都是“Cache-主存-辅存”层次，而且 CPU 以虚地址访问存储系统。若对应虚地址的单元已在 Cache 中，则需把虚地址变换成 Cache 地址，再访问 Cache；若对应虚地址的单元已在主存，但未调入 Cache，则需把虚地址变换成主存地址（经

快表) 和 Cache 地址, 再访问主存, 并将该单元所在块调入 Cache; 若对应单元不在 Cache, 也不在主存, 则需把虚地址变换成辅存实地址、主存地址 (经慢表) 和 Cache 地址, 由辅存将该单元所在页调进主存, 并接着由主存调进 Cache, 对主存进行访问 (与调进 Cache 并行)。虚地址、主存实地址、Cache 地址的对应关系如图 3-54 所示。

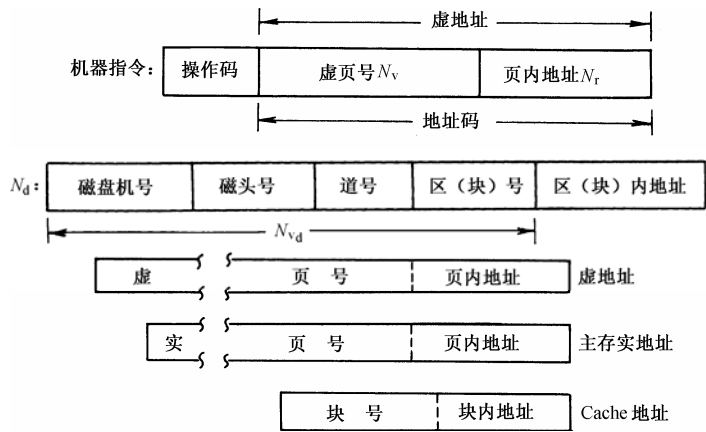


图 3-54 虚地址、主存实地址、Cache 地址的对应关系

由虚地址到主存实地址的变换需经过快表, 如果访问 Cache 是在查取主存实地址后才开始, 势必要在访问 Cache 的周期内增加访问快表的时间。因此, 多数处理方法是查快表和访问 Cache 并行进行。其基本思想是: 在用虚页号去查快表以取得主存实地址的同时, 也用虚地址对应 Cache 块号位置的虚块号经组相联去访问 Cache (Cache 中每个单元包含主存实地址和所对应的数据), 并用由快表取得的主存实地址与由 Cache 读出单元内的主存实地址相比较。相符时, 所在单元的数据即为对应此虚、实地址的 Cache 内容 (即有效信息); 不相符时, 即出现 Cache 不命中, 按既定替换策略对 Cache 进行调块, 同时以主存实地址访问主存的对应单元。写 Cache 过程与此类似。由于每次访问 Cache 都要查快表, 因此, 查快表的速度必须尽可能快, 不能因它而延长访问 Cache 的周期。快表的内容是能随任务切换而变化的, 因此 Cache 内容要能正确反映任务的切换。在实际实现中, 可以有很多方法。例如, Cache 中每个单元所存的是主存实地址的某种压缩编码, 或者以与主存实地址无关的虚地址访问 Cache, 等等。构建多级存储层次会出现许多问题, 如多个虚地址对应同一个实地址等, 但只要掌握前述“主存-辅存”和“Cache-主存”这两级层次的基本概念和分析方法, 问题都是可以解决的。存储体系的设计不能只从自身出发, 还必须联系处理机的要求和操作系统的需求来全面考虑。

3.6 主存保护与控制

3.6.1 主存保护

对于多用户系统和多处理机系统, 其主存同时存有多个用户程序和系统软件。为了使系统能正常工作, 应防止由于一个用户程序出错而破坏主存内的系统软件和其他用户程序, 同时要防止用户程序超出分配给它的主存区域而进行非法访问。为此, 系统结构要为主存的使

用提供存储保护，这是多用户系统和多处理机系统必不可少的。

主存保护包括存储区域的保护和访问方式的保护。为了实现区域保护，对于不具备虚拟存储器的主存系统采用界限寄存器方式，由系统软件经特权指令对上、下界限寄存器置值，从而划定每个用户程序的区域，禁止越界访问。由于用户程序无权改变上、下界限寄存器的值，因此不论它如何出错，只能破坏该用户程序本身，不能侵犯别的用户程序和系统软件。界限寄存器方式只适用于每个用户程序占用一个或几个（当有多对上、下界限寄存器时）连续的主存区域。对于虚拟存储器系统，由于一个用户的各页离散地分布于内存，从而无法使用这种保护方式。虚拟存储器的主存保护常采用页表、键式、环式等保护方式。

1. 页表保护

每个程序都有自己的页表，页表大小（行数）等于该程序的虚页数。若它有 5 页，则只能有 0, 1, 2, 3, 4 五个虚页号，由操作系统建立的该程序页表如表 3-8 所示。

表 3-8 程序页表

虚页号	实页号
0	1
1	3
2	8
3	12
4	18

由表 3-8 可知，不论该程序的虚地址如何出错，只能影响到分配给它的 1, 3, 8, 12, 18 号实页。假如虚页号错成“6”，必然不可能在该页表内找到，其结果是不能访问主存，也就不能侵犯其他程序的区域，这是虚拟存储器系统本身固有的保护机能，也是它的一大优点。为了便于实现，可在段表（可以是页表层次中第一级，不只是段、页式管理中才有）中的每行内设置下一层

次页表的起点和该段的长度（即页数），若出现该段内的虚页号大于段长，则可发出越界中断。页表保护是在形成主存实地址前进行的，使之无法形成侵犯其他程序区域的主存实地址。然而，若由于形成地址的硬、软件出现故障，出现了非法的主存实地址，页表保护无能为力。因此，应采取进一步的保护措施，即键式保护。

2. 键式保护

由操作系统根据当时主存分配情况，给主存的每页配一个键，称为存储键（即关键字，按某种形式组成的编码）。所有页的存储键存在主存相应的快速寄存器内，每个用户（任务）的存储键值是不相同的，但分配给一个用户的所有实页的存储键值是相同的，与存储键对应的是访问键，由操作系统为每个用户确定，存在处理机的程序状态字（PSW）或控制寄存器中。程序每次访问主存前，要核对主存实地址所在页的存储键与该程序的访问键是否相符，只有相符，才许访问。这样，即使错误地形成了侵犯别的程序的主存实地址，也因有键的保护而仍然不许访问。属操作系统的访问键，不论是否与存储键相符都可访问，这是操作系统应能访问整个主存区域所要求的。

3. 环式保护

上述保护方式仅对别的程序区域不受侵犯起保护作用，若要使正在执行的程序本身不被破坏需采取另外的措施，环式保护即是其中之一。环式保护的基本思想是：把系统软件 and 用户程序按其重要性及对整个系统能否正常工作的影响程度分层，如图 3-55 所示。设 0, 1, 2 层是系统软件层，其外侧各层是同一用户程序层。环号表示保护的级别，环号越大，等级越低。在现行程序运行前，由操作系统定好程序各页的环号，并填入页表，而后把该道程序的开始环号填入处理机内的现行环号寄存器，并把操作系统规定该程序所能进入的最内层环号（也称上限环号）同时填入相应的寄存器。设 P_i 是在某一时期属 i 层的各页的集合，当进程执

行 $P \in P_i$ 页内程序时, 若 $j \geq i$, 则允许访问 $F \in P_j$ 页; 若 $j < i$, 则由操作系统的环控制程序判定向内层访问是否合法, 合法时允许访问, 非法时则出错, 进入保护处理。通常 j 值不能小于给定的上限环号。换言之, 只要 $j \neq i$, 就进入中断, 如果允许访问, 就用特权指令把现行环号寄存器内的 i 改为 j 。

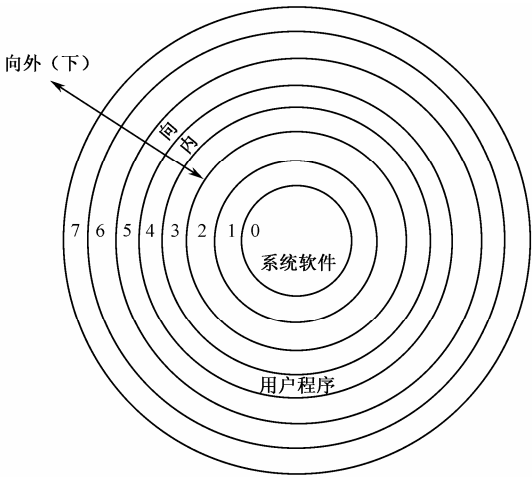


图 3-55 环式保护

环式保护既能保证用户程序出错不至于侵犯系统软件, 也能保护用户程序低级(环号大)部分出错不至于破坏其高级(环号小)部分。环式保护有利于系统软件的研制和调试, 因为可以做到修改系统软件的外层部分而不必担心影响到已调好的核心部分。至于 $j \neq i$ 的跨层访问, 可用规定的转移指令执行, 且对 $j > i$ 和 $j < i$ 分别用不同的转移指令。

上述各种区域保护均由硬件实现, 因此速度很快。区域保护允许对访问的区域进行任何形式的访问, 而对允许区域之外, 任何形式的访问都不允许。但这种限制还不能满足各种应用的要求, 因此, 还需加上访问方式的限制(保护)。

对主存信息操作可有读(R)、写(W)和执行(E)(执行是对指令而言, 相应就有 R, W, E 的访问方式保护)。这三者的逻辑组合可反映各种应用要求。例如:

$\overline{R + W + E}$ ——不允许进行任何访问(如专用的系统表格)。

$R + W + E$ ——可进行任何访问。

$(R + W) \square \overline{E}$ ——只能进行数据读、写, 而不能执行指令(如阵列数据)。

$\overline{R + W} \square E$ ——只能执行, 不能读写数据(如专用程序)。

$R + E \square \overline{W}$ ——只能写访问(如用户对操作系统缓冲寄存器的写入)。

$(R + E) \square \overline{W}$ ——不准写访问。

$\overline{E + W} \square R$ ——只能读访问(如对各用户都要用的表格常数)。

$(E + W) \square \overline{R}$ ——不准读访问。

前述各种区域保护, 都可增加相应的访问方式位以实现各种访问限制。

对界限寄存器方式, 可以在下界寄存器添加一位访问方式位。它为“0”时, 表明该区域可读、可写; 它为“1”时, 表明只准读、不准写。

对键方式，也可添加访问方式位。它的作用可与键值相符一起考虑。如可设计成当“读”保护位为“0”时，不论键值是否相符，均可进行读访问；当“读”保护位为“1”时，只有键值相符才能进行读访问。对于写访问，只有键值相符才能进行。由于写保护而禁写时，被保护的页的内容可保持不变；因读保护而禁读时，被保护的页的内容不能取到寄存器，也不能传送到其他页或 I/O 设备。当然，通过使读保护位为“0”，可实现主存只读部分的多用户共享。

对页表保护方式，可把 R, W, E 等访问方式位设于各道程序的段、页表的各行内，从而使同一段内各页可以有上述各种访问保护，以增强灵活性，而且可以做到各用户对同一共享页的访问方式不同，有的只能读，有的读、写均可。

对环式保护，与页表保护类似，可使同一环内各页有上述各种不同的访问保护，还可以做到使访问保护不仅取决于 R, W, E 的组合，还取决于 j 与 i 值相比之结果（如相等、大于、小于等）。

上述各种访问方式的保护也是由硬件实现的。除此之外，对于多用户共享页可实现只有当前用户访问结束后，别的用户才可访问该页，以防止一个用户还未把共享文件写好之前，别的用户就把它读走了，可利用“测试与置定”和“比较与交换”指令实现这一点。

3.6.2 主存控制部件

由于计算机结构从以 CPU 为中心发展到以主存为中心，存储器形成了体系（层次），因此，对存储器的访问出现了多种形式，也提出了多种要求。例如，虚地址需经过复杂变换才能得到访问主存的实地址；在存储体读/写操作之前需经过存储保护的检验；从存储体读出的信息不能直接送处理机，而需要经过错误检测及错误纠正；对于按字节编址的主存以及采用多体并行交叉结构的主存，需要把存储体读出的信息经过合并或分离之后才能成为处理机所需的信息；对于有 Cache 的存储体系，需要确定虚地址是访问主存，还是访问 Cache；对于多处理机系统，还需协调多个 CPU 和多个 I/O 处理机（通道）对主存的访问。凡此种种都使得主存系统除存储体外，还要有复杂的主存控制部件（存控）来完成这些功能。不同的机器对存控的安排不同，有的把上述某些功能（如主存地址形成和读出信息的合并或分离）由处理机实现。但是，不论放在哪里，上述这些控制功能是必须具备的。

在单 CPU 系统内，访问主存的申请有若干种类，应该按照申请优先级分别处理。优先次序一般如下排列：Cache 访存申请、通道访存申请、CPU 直接对主存写数据申请、CPU 直接对主存读出数据申请、CPU 取指令访存申请等。Cache 申请级别最高，这是因为 Cache 调块时，CPU 空等。至于通道申请级别比 CPU 高，主要原因是通道的缓冲量小，如不及时响应就有可能造成 I/O 信息丢失，而且通道申请中包括取通道控制字和通道地址字，如不及时提供就会造成通道不正常和外设的错误动作。CPU 的写数、读数、取指三种申请，在流水线机器中，由于地址相关问题，要把写数申请置于读数申请之前，使读出的数据确是在它之前应写入的数据；取指申请级别最低是因为取指迟一些不会造成整个机器工作出错，何况采用指令先取技术的机器有指令队列缓冲寄存器，取指可与指令执行重叠，插在主存空闲时取指。对于多处理机系统，应设定每个处理机有一个优先级别，存控有排队控制逻辑，按照各个处理机优先级别安排访存的先后次序，以达到减少访存冲突、提高访存效率的目的。

大多数机器是把访存优先级别定死的，但也有些机器在程序执行过程中动态地改变优先级别。有的机器主存系统只有一个入、出端口，由存控排队控制逻辑决定各个申请的响应次序；

也有的机器有多个入、出端口，各个端口优先级别不同，对每个端口的申请可进一步分级。

随着系统结构的发展，可能把更多的逻辑功能扩散到存控部件。例如，对于“测试与置定”和“比较与交换”等指令，就要求由存控保证对指定单元的读/写操作期间不准别的指令和处理机访问。

对存控所有要求确定之后，可进行存控的具体逻辑设计，在设计时一定要减少由虚地址到存储体地址之间、由存储体输出到处理机相应寄存器之间的级数，尽可能不要由于上述种种要求而使访存时间过分增加。

总之，存控是改善系统吞吐能力、提高系统速度和可靠性的不可缺少的部件，也是存储体系能有效工作的重要保障。

3.6.3 磁盘冗余阵列

磁盘阵列（Disk Array）可以有效提高存储系统的可靠性和性能。磁盘阵列的多个盘驱动器、多个盘片和多个磁头总比一个大驱动器、一个盘片和二个磁头可以更大程度上提高吞吐率和容错能力。它将数据分布到多个磁盘上（即数据带状分布），可以同时访问几个磁盘，自然地提升了吞吐率。可将磁盘阵列中若干磁盘列为校验盘和备用盘，存储文件的备份，从而提高了容错能力。磁盘阵列的缺点是多个设备导致设备故障率提升，但通过增加冗余磁盘可有效提高磁盘阵列的可靠性。这样的系统称为廉价磁盘冗余阵列 RAID（Redundant Array of Inexpensive Disk）。

不同的冗余管理方法其代价和性能各不相同。表 3-9 列出了标准 RAID 级别，每个级别有 8 个数据盘和相应的冗余或校验盘数量。RAID 0 为非冗余磁盘阵列。

表 3-9 RAID 级别列表

RAID 级别	名 称	可接受 最小故障数	数据磁盘 数目（举例）	相应检测 磁盘数目	生产该级别 RAID 产品的企业
0	带状无冗余	0	8	0	广泛使用
1	镜像	1	8	8	EMC, Compaq (Tandem), IBM
2	存储器式 ECC	1	8	4	
3	位交错奇偶校验	1	8	1	Storage Concepts
4	块交错奇偶校验	1	8	1	Network Appliance
5	块交错分布式奇偶校验	1	8	1	广泛应用
6	P+Q 冗余	2	8	2	

（1）带状无冗余（RAID 0）

数据在磁盘阵列中是带状分布（Striping），没有冗余容错功能。带状分布对于软件而言相当于单个大磁盘，简化了存储管理。同时，多个磁盘同时工作，提高了大规模访问的性能，如影视编辑系统。

（2）镜像（RAID 1）

每个数据盘配一个冗余盘，数据写入一个盘，同时也写入对应的冗余盘，因此所有信息都有一个副本。如果一个磁盘发生故障，系统可在对应的镜像盘上找到所需信息。因为使用了两倍的磁盘，RAID 1 的硬件代价是最高的。镜像和带状分布可以组合在一起。设要存储 4

个盘的数据，有 8 个盘可使用。一种方式是以 RAID 1 组织方式，把磁盘组成 4 对，然后把数据带状分布，称为 RAID 1+0 或 RAID 10；另一种方式是以 RAID 0 组织方式，创建两个 4 磁盘系列，然后再镜像写入，称为 RAID 0+1 或 RAID 01。RAID 10 为带状镜像，RAID 01 为镜像带状分布。

(3) 位交错奇偶校验（RAID 3）

RAID 3 用 N 个磁盘作为一个磁盘组存储数据，另外用一个磁盘存放用于数据恢复的冗余信息，而不是每一个磁盘的原始数据都有一个完整的副本。因此，冗余盘减少到原来的 $1/N$ 。读/写操作都是对所有磁盘同时进行的。出现故障时用一个磁盘记录检验信息。RAID 3 用于大数据量的应用，如多媒体和特殊科学代码。用一个冗余磁盘存放所有其他磁盘数据的和，当一个磁盘发生故障时，用校验盘上数据减去其他非故障磁盘数据的和，剩下的数据信息就是所丢失的信息。奇偶校验就是简单的模 2 操作。RAID 3 要求故障率低，优点是只需一个冗余盘，成本较低，缺点是故障恢复时间长。

(4) 块交错奇偶校验（RAID 4）和块交错分布式奇偶校验（RAID 5）

RAID 4 和 RAID 5 与 RAID 3 一样，只用一个冗余盘，但访问数据的方法不同。盘上数据以扇区（块）为单位读/写，奇偶校验也是针对扇区内数据进行差错检测。RAID 3 每次访问都是针对所有磁盘进行的。对于小数据量访问，只要访问的信息在一个扇区内，就允许这些不相干的小数据量读操作并行进行。但是写操作是另一回事，一次小数据量的写操作需要所有其他磁盘读出用于计算奇偶校验的其他信息。设有 4 个数据盘和一个校验盘，如图 3-56 所示，有一块新数据 D_0' 写入（ D_0' 来自 CPU），首先要读出 D_1, D_2, D_3 的数据，与 D_0' 一起进行异或（即计算奇偶），得到新的奇偶校验块 P' ，然后把 D_0' 写入，再把 P' 写入。

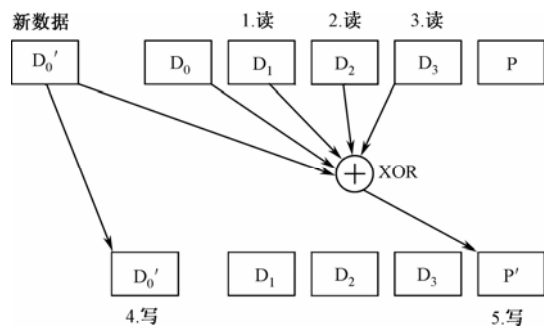


图 3-56 RAID 3 数据写入

RAID 4 和 RAID 5 写入与 RAID 3 不同，如图 3-57 所示。先读出旧块 D_0 与新块 D_0' 比较，确定哪些位改变，然后读取旧校验块 P 并改变相应位成为 P' ，最后写入 D_0', P' 。当然比较仍然采用异或 (XOR)。RAID 4 和 RAID 5 用读两块 (D_0, P) 写两块 (D_0', P') 代替 RAID 3 读三块 (D_1, D_2, D_3) 写两块 (D_0', P')。RAID 4 和 RAID 5 只对两个磁盘 (D_0, P) 进行操作。

奇偶校验只是信息的按位和（异或），即

$$\begin{aligned}
 \text{偶校时, 校验位} &= b_{n-1} \oplus b_{n-2} \oplus \cdots \oplus b_1 \oplus b_0 \\
 \text{奇校时, 校验位} &= b_{n-1} \oplus b_{n-2} \oplus \cdots \oplus b_1 \oplus b_0 \oplus 1
 \end{aligned}$$

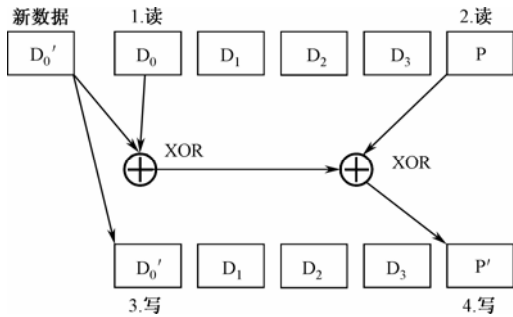


图 3-57 RAID 4 和 RAID 5 数据写入

所以，分析写入新数据后哪些位发生变化，然后改变校验盘上的对应位即可。图 3-57 说明了这种方法，通过读出旧数据，用旧数据与新数据比较，判断哪些位发生变化，再读旧校验和，改变对应有变化的位，最后写入新数据和新的校验和。因此，一次小数据量的写操作包含对两个磁盘的 4 次访问，而不是访问所有的磁盘（RAID 3），这种结构就是 RAID 4。RAID 4 可以有效支持大数据量读、小数据量读、大数据量写、小数据量写的混合操作；缺点是每次写操作都要更新校验盘，这是顺序写操作的瓶颈。为了打破这种瓶颈，RAID 5 采用将校验信息分布到所有其他盘上，使写操作不存在单一的瓶颈。图 3-58 说明 RAID 4 和 RAID 5 数据分布。RAID 5 校验信息不再限于某个磁盘，只要写的信息不同时位于同一个磁盘中，就可以使多个写操作并行进行。例如，第一个写操作访问第 8 块，需要同时访问 P_2 。第二个写操作访问第 5 块，需要更新 P_1 。对于 RAID 4，由于 P_1 、 P_2 同在第 5 个盘上，写第 5 块和写第 8 块不能同时进行，出现了瓶颈。而对于 RAID 5，第 5 块在第 2 个盘上， P_1 在第 4 个盘上；第 8 块在第 1 个盘上， P_2 在第 3 个盘上，相互无干扰，两个操作可以并行进行。

（5）P+Q 冗余（RAID 6）

奇偶校验只能发现奇数个错误，对于偶数个错误不能发现。RAID 6 则是在一次奇偶校验的基础上，对数据和校验盘（P）的信息进行第二次奇偶校验，形成第二个校验盘（Q）。二次校验机制可使系统从二次故障中鉴别出错误。二次校验比一次复杂，因此 RAID 6 代价是 RAID 5 的两倍。图 3-56 所示数据写操作，对 RAID 6 需要 6 盘同时访问，并改写 P 盘和 Q 盘。

RAID 具有较高吞吐率及故障恢复能力，同时，盘径小，驱动器体积小，耗能低。RAID 在大型存储系统中是非常重要的环节，得到广泛应用。在 RAID 设计时，要解决三个问题：

- （1）磁盘故障记录。盘的扇区上记录了该扇区是否发生故障的信息，在读磁盘时，电子设备可通过这些信息发现磁盘故障或信息丢失的情况。
- （2）减少平均修复时间（MTTR）。典型方法是在系统中加入热备件（Hot Spare），就是在正常情况下不使用的磁盘。当 RAID 出现故障时，热备件启动，将其他磁盘中的冗余数据（即故障盘数据备份）恢复到热备件中。该过程是自动进行的，所以平均修复时间将大大降低。
- （3）热交换（Hot Swapping）。在不中断计算机工作（不脱机）的前提下，更换 RAID 中损坏的组件，丢失的数据可以立即填入备件中。

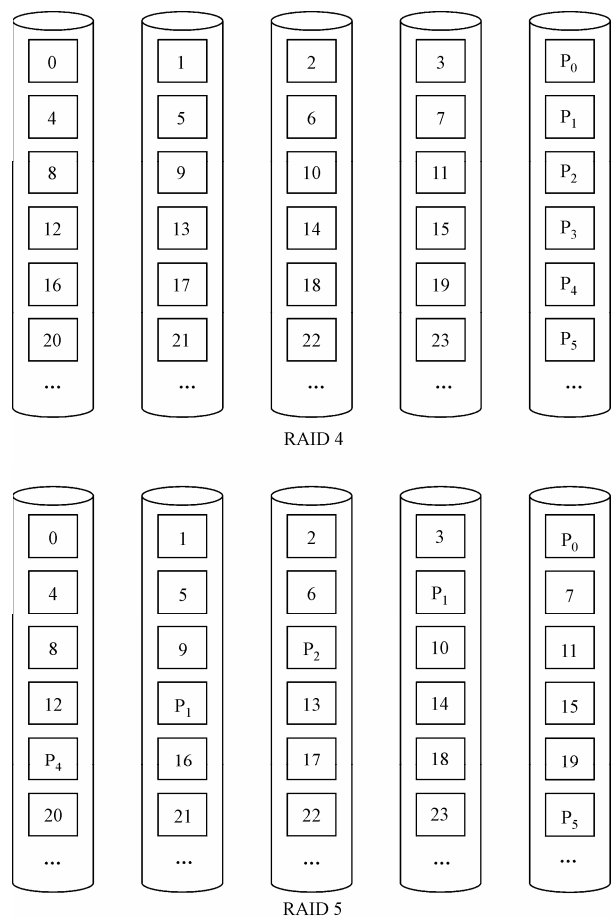


图 3-58 RAID 4 和 RAID 5 数据分布

第 4 章 流水线结构

4.1 流水线结构原理

如何加快指令的解释过程是计算机系统设计的基本任务。除了采用高速部件外，一次重叠、先行控制和流水等控制方式是常用的，意在提高指令的并行性，从而加速指令的解释过程。

流水技术不只用于指令的解释过程，还广泛应用于计算机系统结构的其他方面，如向量的流水处理。

4.1.1 重叠方式

1. 重叠方式原理

一条指令的解释过程可以根据完成这个过程的微操作分为若干子过程。如一般讨论时，分成取指和分析执行两个子过程或取指、分析和执行三个子过程。如图 4-1 所示。取指子过程是把要处理的指令从存储器里取到处理机的指令寄存器；分析子过程则包括对指令的译码，形成操作数有效地址并取操作数，形成下一条指令地址；执行子过程是对操作数进行运算并存好结果。

取指分析子过程在指令分析器里完成，执行子过程在执行部件实现。而这两个部件是独立的，如图 4-2 所示。可以设想，在上一条指令的分析子过程在指令分析器中结束，并将结果送入执行部件去实现执行子过程时，指令分析器不必等本指令在执行部件完成有结果后再对下一条指令进行分析子过程，而是同时进行。如果分析子过程所需的时间（即分析周期）为 Δt_1 ，执行子过程所需时间即执行周期，与分析周期相同，那么执行部件里处理的是第 n 条指令的执行子过程，而分析器里处理的是第 $n+1$ 条指令的分析子过程，如图 4-3 所示。这就是一次重叠技术，它体现了一定的并行性。从顺序处理看，每条指令需 $T=2\Delta t_1$ 时间，而一次重叠则每个 Δt_1 时间就完成一条指令的处理，使处理机的速度提高一倍。



图 4-1 指令的解释

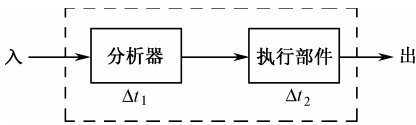


图 4-2 指令分解为两个部件来实现

如果取指、分析和执行时间相等，设均为 t ，则采用顺序方式解释 n 条指令所需时间为

$$T_0=3nt$$

如果时间不等，则所需时间为

$$T_0=\sum_{i=1}^n (t_{取指_i}+t_{分析_i}+t_{执行_i}) \quad (i=1, 2, \cdots, n)$$

按照图 4-3 所示一次重叠处理方法，把执行和分析重叠，假设各机器周期时间相等则解释同样的 n 条指令所需时间为

$$T_1=(1+2n)t$$

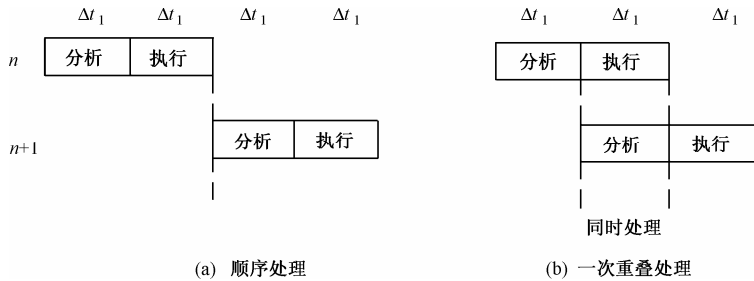


图 4-3 顺序处理与一次重叠处理

时间减少近 1/3。这不但使主存一直处于工作状态，而且使其他部件利用率也得到提高。但为了实现重叠方式，处理机必须增加一个指令缓冲寄存器，在执行第 k 条指令时，存储取来的第 $k+1$ 条指令。如果进一步增加重叠，如图 4-4 所示（图中 $k, k+1, \dots$ 表示第 k 条，第 $k+1$ 条指令 \dots ），把“分析”与“取指”重叠起来，则处理机速度还可以进一步提高。仍假定各机器周期时间相等，则按这种方式解释 n 条指令所需时间为

$$T_2=(2+n)t$$

所需时间减少近 2/3，这是一种理想的重叠方式，它加快了相邻两条指令以至一段程序的解释。



图 4-4 一次重叠的第二种方式

2. 重叠方式结构

为了实现重叠方式，处理机组上有相应的结构。图 4-5 为一次重叠方式的基本结构。

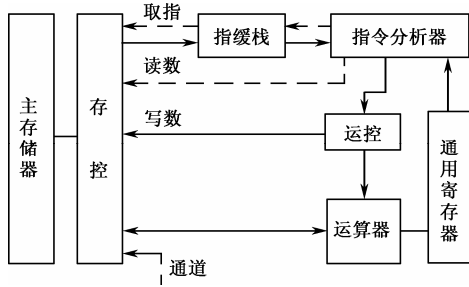


图 4-5 一次重叠方式的基本结构

首先，为了在“执行 k ”的同时，能“分析 $k+1$ ”和“取指 $k+2$ ”，就需要有独立的取指令部件、指令分析部件和指令执行部件。它们有各自的控制线路，实现各自的功能。为此，

把处理机中原来一个集中的控制器，分解为存储控制器（存控）、指令控制器（指控）和运算控制器（运控）三个部分。

其次，还要解决访存冲突（Collision）。“分析 k ”要访存取操作数，“取指 $k+1$ ”也要访存取指令，在一般机器中，操作数和指令是存储在同一主存内，而主存在同一时刻只能访问一个存储单元。这样就形成了“分析 k ”与“取指 $k+1$ ”的访存冲突。也就是在这一步不能实现重叠。

有三种解决方法：

（1）使操作数和指令分别存于两个独立编址且可同时访问的存储器内。显然，这种方法对程序员是不透明的。

（2）采用多体交叉存储器。这时，操作数和指令仍能够混存。但是如第 k 条指令所需操作数与第 $k+1$ 条指令不在一个存储体内，则可同时取到两者，实现“重叠”。如正好在同一体内，仍无法实现“重叠”。

（3）设置指令缓冲寄存器组（指缓栈），如图 4-5 所示。把所需的后继指令预先取到指缓栈（指令先取队列），则“分析 k ”与“取指 $k+1$ ”就能够重叠。因为前者访存是取操作数，后者到指缓栈取下一条指令，互不干扰。有的指令在“分析”时不需取操作数，有的指令在“执行”时所需时间又比“分析”时间长，因此主存总会有空闲时间。利用这段时间，可以把指令预先取进队列中。指缓栈的基本组成如图 4-6 所示。它除了有一组指令缓冲寄存器外，还有自己的控制逻辑，它按“先进先出”的方式工作，保证指令顺序不乱。原则上，只要指令缓冲寄存器没有满，就可以自动向存控发出取指请求。指令分析器完成一条指令的分析，就向指令缓冲寄存器求取指令。由于主存读取指令速度与指令分析器取走指令速度不一样，因此队列中存放的指令条数是动态的。具有指令缓冲寄存器的处理机有两个程序计数器：一个是程序先行计数器 PC_1 ，用于向内存取指；另一个是现行程序计数器 PC ，用于记录指令分析器正在分析的指令地址。

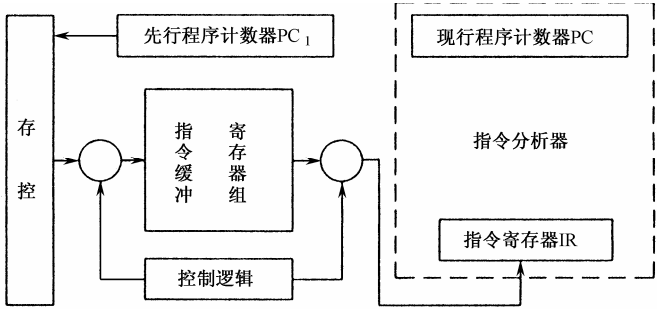


图 4-6 指令缓冲队列基本组成

若每次“取指”的时间很短，则可将“取指”合并并在“分析”周期内，从而构成如图 4-4 所示一次重叠方式。一次重叠方式连续解释 n 条指令时，其所需时间为

$$T_3=(1+n)t$$

由于程序中不可避免地会出现“转移”、“调子程序”和各种中断，也因此会改变程序的顺序流程。而各种类型指令的“分析”和“执行”周期不尽相同。所以， T 是一种难以到达的、理想的境界。

4.1.2 先行控制

1. 先行控制原理

一次重叠方式进行时，都是“执行 k ”与“分析 $k+1$ ”重叠。如果有的指令“分析”与“执行”的时间均相等，则一次重叠的流程非常流畅，无任何阻碍，机器的指令分析部件和执行部件的功能充分地得到发挥，机器的速度也能显著地提高。但是，现代计算机的指令系统很复杂，各种类型的指令难以做到“分析”与“执行”时间始终相等。此时，一次重叠流程中可能出现如图 4-7 所示的情况。当“分析 $k+1$ ”结束后，指令分析部件要等待“执行 k ”完成，才能接着“执行 $k+1$ ”，进行“分析 $k+2$ ”。这就出现了“分析 $k+1$ ”时间小于“执行 k ”的情况。又如，“执行 $k+2$ ”时间小于“分析 $k+3$ ”时间，又出现了执行部件等待的情况。这样，指令分析部件和执行部件就不能连续、流畅地工作，从而机器的整体速度受到影响。图 4-7 中 Δt_1 和 Δt_2 为指令分析部件流程中等待时间； Δt_3 为执行部件流程中的等待时间。

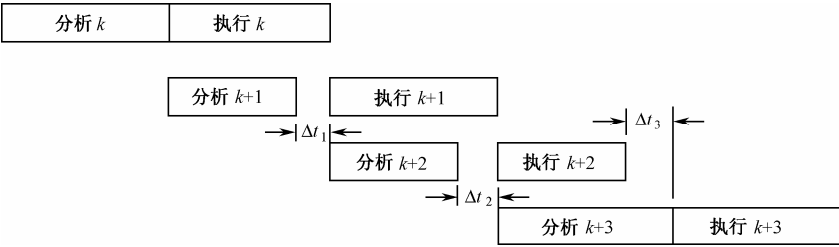


图 4-7 “分析”和“执行”时间不等的一次重叠

在只有一套指令分析部件和执行部件的前提下，如何减少这种情况下机器速度的损失呢？其基本思想是：在执行部件执行第 k 条指令的同时，指令控制部件能对其后继的第 $k+1$ ， $k+2$ ， \cdots 条指令进行预取和预处理，为执行部件执行新的指令做好必要和充分的前期准备。这样，就能使指令分析部件和指令执行部件连续、流畅地工作，流程中指令分析部件和执行部件之间有等待的时间间隔 Δt 如图 4-8 所示，但它们各自流程却是连续的。这种方式称为先行控制方式，其时序如图 4-8 所示。

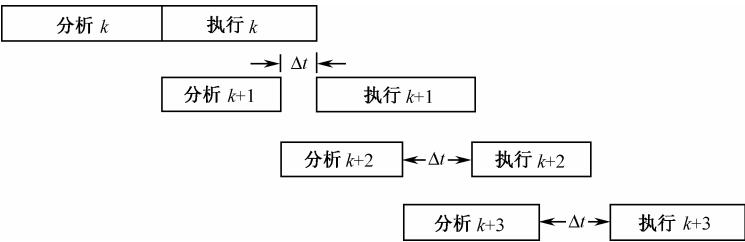


图 4-8 先行控制方式的时序

采用先行控制方式，在理想的情况下，连续解释 n 条指令所需时间为

$$T_{\text{先行}} = T_{\text{分析 } 1} + \sum_{i=1}^n T_{\text{执行 } i} \quad (i=1, 2, \cdots, n)$$

2. 先行控制结构

采用先行控制方式的处理机结构如图 4-9 所示。在指令控制部件中，除了原有指令分析

器外，又增加了先行指令栈、先行读数栈、先行操作栈和后行写数栈。

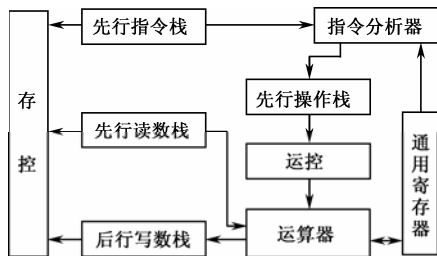


图 4-9 先行控制处理机基本结构

现代计算机组成中，缓冲部件使用较多，它们一般设置在两个工作速度不同的部件之间，起到平滑其工作速度的作用。缓冲技术是计算机组成设计的一个重要技术。

先行指令栈的作用是后继指令预取，保证指令分析器在顺序取指时能从先行指令栈内取到，它相当于一次重叠结构中的指缓栈（如图 4-5 所示）。所以，先行指令栈是主存与指令分析器之间的一个缓冲部件，用于平滑主存和指令分析器之间的工作速度。当指令分析器分析某条指令用时较长或主存空闲时，可多取几条指令存入先行指令栈，以作备取。

指令分析器完成指令译码后，经过寻址操作得到操作数有效地址。如果仍由指令分析器向存控发取数请求信号，则必然等待存控的响应，这就妨碍了后继指令的连续处理。若将有效地址送先行读数栈内的先行地址缓冲寄存器，则指令分析器可以继续处理后继指令。先行读数栈由一组先行地址缓冲寄存器、先行操作数缓冲寄存器和相应的控制逻辑组成。每当地址缓冲寄存器接到有效地址时，控制逻辑主动向存控发取数请求信号，读出的数据送到先行数据缓冲寄存器内。先行读数栈以“先进先出”的方式工作。运算器直接从其中读取数据操作，不向主存取数。所以先行读数栈是主存和运算器间的缓冲部件。先行读数栈内的数据对运算器内正在执行的指令而言属于后继指令执行所需的数据，故称“先行”。指令分析器对指令预处理，将原来交给运算器执行的各种机器指令转换成运算器能够执行的操作命令，从而把运算器原来的访存操作变成访问先行操作缓冲器。

指令分析器与运算器之间的缓冲部件是先行操作栈。由一组操作命令缓冲寄存器及相应控制逻辑组成。指令分析器预处理完一条指令，将相应操作命令送入先行操作栈，指令分析器继续对后继指令进行预处理。而运算器通过运控从栈内按顺序逐个取出操作命令执行。同样，先行操作栈内命令对于运算器内正在执行的命令而言是“先行”的。

对于“写数”命令，则需将有效地址送入后行写数栈中的后行地址缓冲器。运算器执行“写数”操作命令，只需将数据送入后行写数缓冲寄存器即可，不与主存打交道，可继续执行后继命令。后行写数栈由一组后行地址缓冲寄存器、后行写数缓冲寄存器及相应控制逻辑组成。每当接到运算器送来的要写入主存的数据时，由控制逻辑自动向主存发写数请求信号，完成存数操作。它是运算器和主存间的缓冲部件。由于后行写数栈中写回的数据，对于运算器中正在执行的命令而言，是先前命令“滞后”写回的数据，故叫后行写数栈。它也是按“先进先出”的方式工作。

综上所述，先行控制技术实际上是缓冲技术和预处理技术相结合的产物。通过对指令流和数据流的先行控制，尽量使指令分析部件和执行部件处于忙碌状态。与一次重叠相比，其不同之处在于，指令分析部件和执行部件可以同时处理两条不相邻的指令，即实现多条指令

重叠解释，因此它的并行性更高。通常把先行指令栈、先行读数栈、先行操作栈和先行写数栈统称为先行控制器。它与指令分析器一起构成先行控制方式中的指令分析、控制部件，而运算器及其运控构成了执行部件。

4.1.3 流水线处理机

1. 流水技术原理

在一次重叠方式中，把一条指令解释过程分解为“分析”与“执行”两个子过程，每个子过程需时 Δt_1 （如图 4-2、图 4-3 所示）。

如果把解释过程进一步细分为“取指”、“译码”、“取操作数”和“执行”4 个子过程，并分别由各自独立的部件来实现。显然，按顺序处理，每条指令需 $T=4\Delta t_2$ 时间，而现在则是每一个 Δt_2 流出一条指令的处理结果。这种工作方式类似于现代工厂的装备流水线，每隔 Δt_i

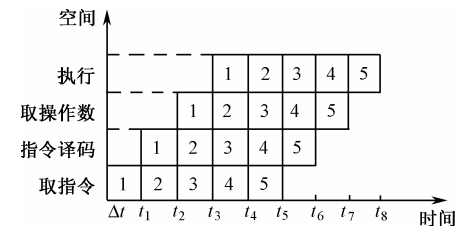


图 4-10 流水线技术原理时空图

连续处理 5 条指令的流水过程。时空图的横坐标表示时间，纵坐标表示流水线的各功能段。

- 一般而言，流水技术有如下特点：
- （1）可以划分为若干互有联系的子过程（功能段）。每个功能段有专用功能部件实现。
 - （2）实现子过程的功能段所需时间尽可能相等，避免因不等产生处理的瓶颈，形成流水线“断流”。
 - （3）形成流水处理，需要一段准备时间，称为“通过时间”。只有在此之后流水过程才能够稳定。
 - （4）指令流不能顺序执行时，会使流水过程中断，再形成流水过程，则需通过一段时间。所以流水过程不应常“断流”，否则效率不会很高。
 - （5）流水技术适用于大量重复的程序过程，只有输入端能连续地提供服务，流水线效率才能够充分发挥。

2. 流水结构分类

流水结构不仅在指令的解释过程中可以提高处理速度，而且可以应用于各种大量重复的时序过程，如浮点数加法器等。因此它有按不同结构和不同观点的分类。

- （1）按完成的功能分类
- ① 单功能流水线。只能完成一种功能的流水线，如只能实现浮点数加法、减法或乘法。在计算机中要实现多个功能，都采用多个单功能流水线，如图 4-11 所示。

时间流水线就流出一个产品，而制造一个产品的时间远远大于 Δt_i ，这就是计算机设计中的“流水线”概念。

流水线技术将一个重复的时序过程分解为若干子过程，而每个子过程都可以有效地在其专用功能上与其他子过程同时执行。

描述流水线的工作过程，常采用时（时间）空（空间）图的方法。图 4-10 是 4 个解释子过程

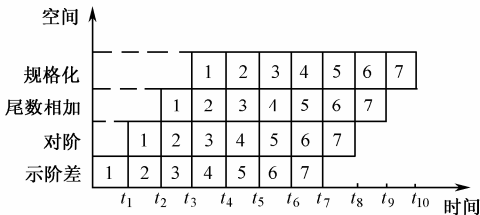


图 4-11 浮点数加法单功能流水线时空图

② 多功能流水线。同一个流水线可用多种连接方式来实现多种功能，一个典型的例子是 ASC 运算器，它有 8 个功能段，经适当连接可以实现浮点数加/减或定点乘等功能，如图 4-12 所示。

(2) 按同一时间内各段之间的连接方式分类

① 静态流水线。同一时间内，流水线的各段只能按同一种连接方式工作，如图 4-13 (a) 所示，只能按浮点数加法工作或定点数乘法工作。

② 动态流水线。在同一时间内，流水线的各段可按不同运算的连接方式工作。如图 4-13 (b) 所示，有些段实现浮点数加，有些段实现定点数乘，显然工作效率提高了，但控制复杂多了。因此，大多数流水线都是静态的。

(3) 按流水的级别分类



图 4-12 ASC 运算器多功能流水线

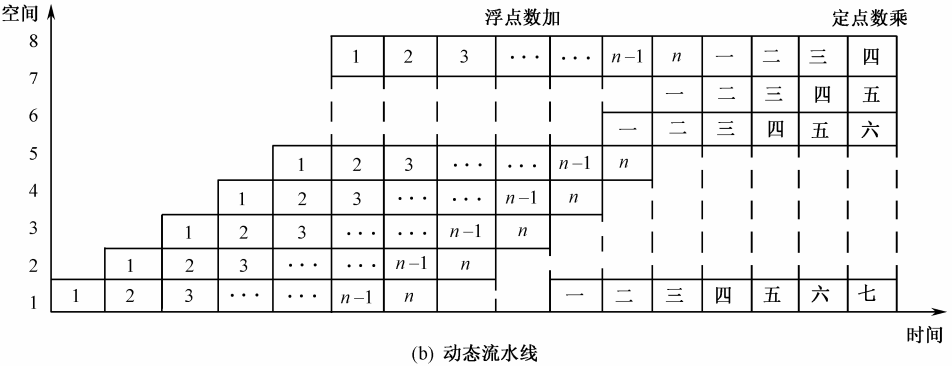
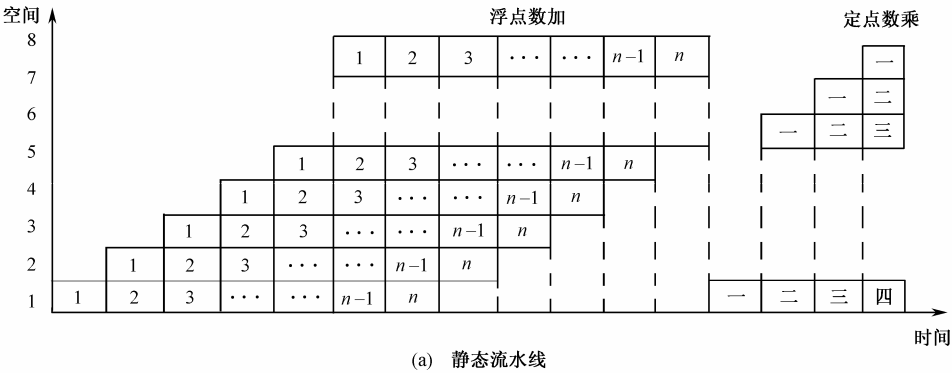
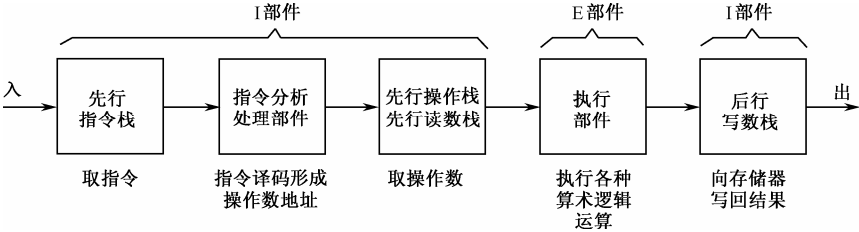


图 4-13 静、动态流水线时空图

① 部件级流水线。又称运算器流水线，它是按处理机的算术逻辑部件分段，使各种数据类型能进行流水操作。就像前面图 4-11 和图 4-12 所举的例子。

② 处理机级流水线。又称指令流水线，它是在指令解释过程中划分为若干功能段，按流水方式组织起来，就像前面图 4-10 所举的例子。先行控制也是指令流水线，它把指令过程分为 5 个子过程，用 5 个专用功能段进行流水处理，见图 4-14 所示。



如果把解释过程分成两个子过程，“分析”和“执行”的“一次重叠”，就是最简单的指令流水。

③ 处理机间流水线。又称宏流水线，它是指两台以上的处理机串行地对同一数据流进行处理，每台处理机完成一个任务。这种结构往往又称为异构型多处理机系统，如图 4-15 所示。

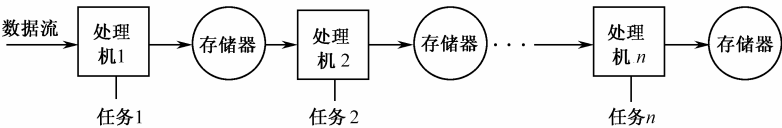


图 4-15 宏流水线

(4) 按数据表示分类

- ① 标量流水处理机。它只能对标量数据进行处理。
- ② 向量流水处理机。它具有向量指令，能对向量的各个元素进行流水处理。

(5) 按流水线中是否有反馈回路来分类

- ① 线性流水线。流水线各段串行连接，没有反馈回路。
- ② 非线性流水线。流水线中除了串行连接通路外，还有反馈回路。在流水过程中，有些段要反复多次使用。它常用于递归或组成多功能流水线，如图 4-16 所示。

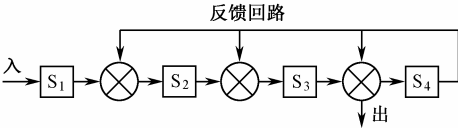


图 4-16 非线性流水线

3. 流水线结构

(1) 运算操作流水线结构

按照指令的要求，选择合适的算法，把运算过程分为多个子过程，使各个子过程的时间尽量相等。但由于实际执行的误差，可能引起各段之间的干扰。为避免干扰，保证各段之间数据通路宽度匹配，在各段之间增加锁存器（Latch），如图 4-17 所示。因此，锁存器成了各

段之间的标志。所有段与一个统一的时钟同步，经常采用集中控制方式。这种方式的基本问题是如何分段，从而确定时钟周期的大小。

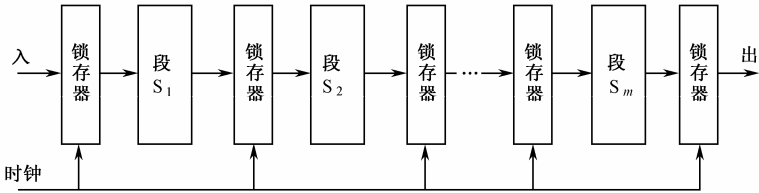


图 4-17 运算操作流水线结构

(2) 指令流水线结构

“一次重叠”和“先行控制”均是指令流水线。但是，在流水线处理机中，由于运算部件采用流水处理，高速时每个节拍可得到一个结果，因此需要进一步提高指令部件的速率，使之每个节拍输出一条指令给执行部件，指令流水过程要分成更多的子过程。由于在指令流水过程中要不断访问存储器，而所需时间又不固定，因此难以用统一的时钟控制各段工作，指令流水线基本结构如图 4-18 所示。该结构在指令处理部件中采用了流水线方式，指令缓冲器（先行指令栈）最多可预取 8 条双倍字长指令，在流水线刚启动时，执行还没开始，不需要取操作数，此时存储器空闲，所以预取 5 条双字长指令。在流水线工作稳定后，指令处理部件不断从存储器取指令，同时将指令从指令缓冲器送到指令寄存器，并进行指令译码。经译码后，预处理过的指令分别送定点、浮点执行区，或者送存控。地址加法器用于寻址操作时的运算，也用于产生转移指令的目标地址。由于条件转移和中断对流水线效率影响较大，因此专门设置了一个控制部件来处理取指令、转移和中断。同时还设置了两个转移目标缓冲器，接收转移目标地址。在中断时，则接收程序状态字（PSW），为中断响应、处理做好准备。

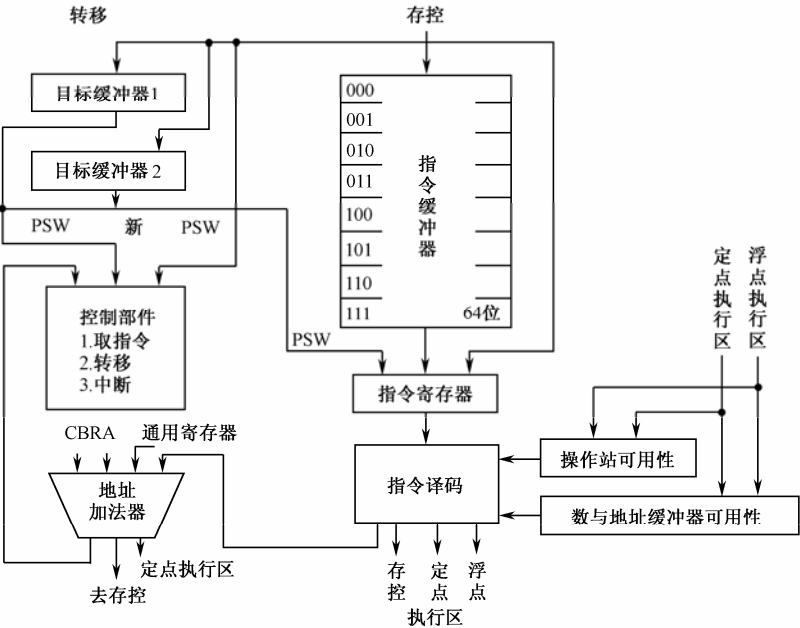


图 4-18 指令流水线基本结构

(3) Pentium 流水线结构

Pentium 微处理器是 Intel 公司的 32 位处理器产品。以 Pentium 为核心构成的计算机系统有相当大的市场份额。Pentium 有两条流水线，即“U”流水线和“V”流水线，U 和 V 都可以执行整数指令，但只有 U 流水线执行浮点指令。在 V 流水线也可以执行一条异常的 FXCH 指令，所以 Pentium 在每个时钟周期内执行两条整数指令，或执行一条浮点指令，如果两条浮点指令中有一条是 FXCH 指令，则一个时钟可以执行两条浮点指令。每条流水线都有自己独立的地址生成逻辑、算术逻辑部件和 Cache 接口。Pentium 有 8KB 指令 Cache 和 8KB 数据 Cache，数据 Cache 有两个端口，分别用于 U 和 V 两条流水线。有一个专用的转换后援缓冲寄存器（TLB），用于把线性地址转换成数据 Cache 的物理地址。指令 Cache、转移目标缓冲器和预取缓冲器将原始指令送入 Pentium 执行部件。指令取自指令 Cache 或外部总线（即外部 Cache 或主存）。指令 Cache 内的 TLB 将线性地址转换成指令 Cache 所用的物理地址。遇到转移时，转移地址由转移目标缓冲器予以记录，译码部件将预取指令处理成 Pentium 流水部件可执行的命令，使 ROM 直接控制两条流水线，内有 Pentium 系统结构能执行的微程序。Pentium 属于超级标量系统结构。

4.2 线性流水线技术指标

流水线的主要性能包括吞吐率、加速比和效率，它们是评价处理机的重要指标。

4.2.1 吞吐率

吞吐率（Throughput Rate）指单位时间内流水线能够处理的任务数（或指令数）或流水线能输出的结果的数量，它是衡量流水线速度的主要性能指标。

流水线在连续流动达到稳定状态后得到的吞吐率称为最大吞吐率，设流水线各功能段时间 Δt_i 都相等，即 $\Delta t_i = \Delta t_0$ ，则流水线的最大吞吐率

$$TP_{\max} = \frac{1}{\Delta t_0}$$

当流水线各功能段时间不等时 $TP_{\max} = \frac{1}{\max\{\Delta t_i\}}$

显然，最大吞吐率取决于流水线中最慢的那个功能段，又称它为“瓶颈”。

借助时空图来分析吞吐率有利于得到实际的吞吐率指标，图 4-19 表示各功能段时间相等的流水线时空图，设流水线由 m 段组成，要完成 n 个任务，所需时间 T 为

$$T = n \cdot \Delta t_0 + (m-1) \cdot \Delta t_0 \quad \text{或} \quad T = m \cdot \Delta t_0 + (n-1) \cdot \Delta t_0$$

实际吞吐率

$$TP = \frac{n}{T} = \frac{n}{m \cdot \Delta t_0 + (n-1) \cdot \Delta t_0} = \frac{1}{\Delta t_0 \cdot \left(1 + \frac{m-1}{n}\right)} = \frac{TP_{\max}}{1 + \frac{m-1}{n}}$$

所以，实际吞吐率小于最大吞吐率，它除了与 Δt_0 有关，还与段数 m 、任务数 n 有关，只当 $m \ll n$ 时，吞吐率 TP 才能接近于 $TP_{\max} = 1/\Delta t_0$ 。

如果各段时间不等，完成 n 个任务的实际吞吐率

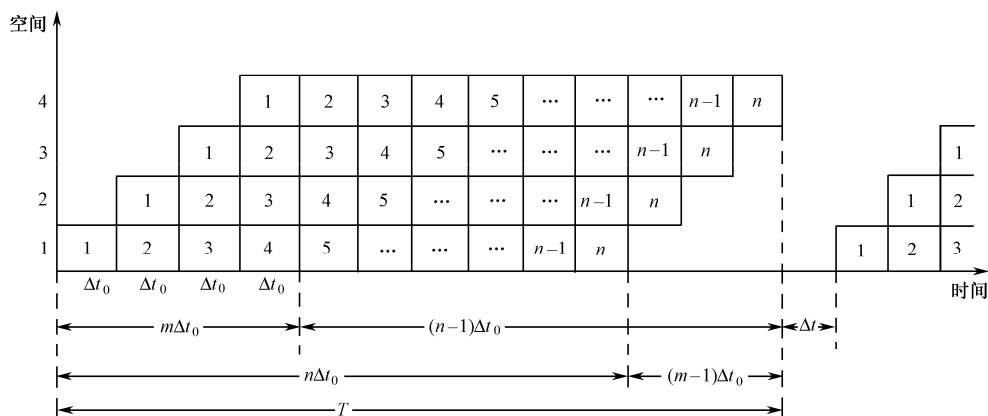


图 4-19 从时空图分析吞吐量

$$TP = \frac{n}{\sum_{i=1}^m \Delta t_i + (n-1)\Delta t_j}$$

式中, Δt_j 为最慢的一段时间。

4.2.2 加速比

m 段流水线的速度与等效的非流水线的速度之比称为加速比 (Speedup Ratio)。

设在各段时间相等的流水线上完成 n 个任务的时间是 $T_{\text{流水}} = m \cdot \Delta t_0 + (n-1) \cdot \Delta t_0$, 而非流水线所需的时间为 $T_{\text{非流水}} = m \cdot n \cdot \Delta t_0$, 所以加速比

$$S = \frac{T_{\text{非流水}}}{T_{\text{流水}}} = \frac{m \cdot n \cdot \Delta t_0}{m \cdot \Delta t_0 + (n-1) \cdot \Delta t_0} = \frac{mn}{m+n-1} = \frac{m}{1 + \frac{m-1}{n}}$$

由此可见, 只有当 $n \gg m$ 时, 加速比 S 才接近于段数 m 。也就是说, 流水线的段数越多, 则加速比就越高, 但此时的任务数 n 也要求越大。

如果各段的时间不等, 则有

$$S = \frac{n \cdot \sum_{i=1}^m \Delta t_i}{\sum_{i=1}^m \Delta t_i + (n-1)\Delta t_j}$$

式中, Δt_j 为最慢的一段时间。

4.2.3 效率

流水线上各段有通过时间和排空时间, 即并不都是满负荷工作的, 流水线上的设备利用率就是效率 (Efficiency)。

设在各段时间相等的流水线上, 每段的效率 e_i 是相等的, 即

$$e_0 = e_1 = \dots = e_m = \frac{n \cdot \Delta t_0}{T} = \frac{n}{m + (n-1)}$$

整个流水线的效率

$$E = \frac{e_1 + e_2 + \cdots + e_m}{m} = \frac{me_0}{m} = \frac{m \square n \square \Delta t_0}{m \square T}$$

式中，分母是时空图中 m 个段和 T 时间所围成的总面积，而分子是 n 个任务实际占用的面积（在图上常用阴影表示）。

如果各段时间不等，则各段的效率也不等，则整个流水线的效率

$$E = \frac{n \square \sum_{i=1}^m \Delta t_i}{m \left[\sum_{i=1}^m \Delta t_i + (n-1)\Delta t_j \right]} = \frac{n \text{ 个任务占用的时空区}}{m \text{ 个段总的时空区}}$$

式中， Δt_j 为瓶颈段的执行时间。

流水线的效率与吞吐率之间一般为正比关系。

$$E = \frac{n \square \Delta t_0}{T} = TP \square \Delta t_0 = \frac{n}{m + (n-1)} = \frac{1}{1 + \frac{m-1}{n}}$$

所以，当 $n \gg m$ 时， $E=1$ 。根据上式加速比 S 公式可知， $E=S/m$ 或 $S=mE$ ，效率 E 是实际加速比 S 和最大加速比 m 之比，只有 $E=1$ 时， $S=m$ ，实际加速比达到最大。

如果流水线的各段时间不等，各段效率也不等，从上式可知

$$E = TP \square \frac{\sum_{i=1}^m \Delta t_i}{m}$$

$$TP = E \square \frac{m}{\sum_{i=1}^m \Delta t_i}$$

由于除了最慢段以外，其他段都出现空白区，效率 E 是较低的，对吞吐率 TP 影响较严重（因为 $m / \sum_{i=1}^m \Delta t_i < 1$ ， $i=1, 2, 3, \cdots, m$ ）。

在实际计算流水线效率时，还需考虑各段设备量不等的情况。根据每段设备量在总设备量中的比例，分别赋予不同的权值 α_i ，此时流水线效率

$$E = \frac{n \text{ 个任务占用加权时空区}}{m \text{ 个段总的加权时空区}} = \frac{n \square \sum_{i=1}^m \alpha_i \Delta t_i}{\sum_{i=1}^m \alpha_i \left[\sum_{i=1}^m \Delta t_i + (n-1)\Delta t_j \right]}$$

式中， $\alpha < m$ ，且 $\sum_{i=1}^m \alpha_i = m$ （ $i=1, 2, 3, \cdots, m$ ）。

对于非线性流水线和多功能流水线，也可仿照上述对线性流水线性能的分析方法，在正确画出时空图的基础上，分析其吞吐率和效率。

4.2.4 流水线段数选择

增加流水线的段数，流水线的吞吐率和加速比都能提高。由于在每段的输出端都必须设

置一个锁存器(或称缓冲寄存器),因此段数增多时各锁存器的延迟时间累加值也将增加,甚至有可能超出流水线各段的延迟时间的累加值。同时,增加锁存器也增加了流水线硬件价格。所以,在设计流水线时,要综合各方面的因素,根据最佳性能价格比的要求,选择流水线的最佳段数。

设在非流水线的机器上顺序执行一个任务所需时间为 t , 在同等速度的有 K 段流水线的机器上执行一个任务所需时间为

$$\frac{t}{K} + d$$

式中, d 为锁存器的延迟时间。流水线的最大吞吐率

$$TP_{\max} = \frac{1}{\frac{t}{K} + d}$$

流水线的总价格估计为

$$C = a + bK$$

式中, a 为所有功能段价格之和, b 为每个锁存器的价格。A.G.Larson 把流水线的性能价格比 PCR 定义为

$$PCR = \frac{TP_{\max}}{C} = \frac{1}{\frac{t}{K} + d} \cdot \frac{1}{a + bK} = f(K)$$

通过对 K 求导,得到 PCR 极值,如图 4-20 所示。当 PCR 取得最大值时,所对应的段数就是最佳段数 K_0 :

$$K_0 = \sqrt{\frac{t \square a}{d \square b}}$$

式中, t 为流水线总的延迟时间。 K_0 与 $\sqrt{t \square a}$ 成正比,与 $\sqrt{d \square b}$ 成反比。在流水线总延迟时间 t

一定的前提下,通过调整流水线本身价格 a 、锁存器延迟时间 d 和锁存器价格 b 来选取最佳流水线段数 K 。目前,一般处理机中流水线段数在 4~10,极少超过 15 段。我们把 $K \geq 8$ 的流水线称为超级流水线。

对于单功能、线性流水线、输入任务是连续的情况,可以通过上述有关公式计算 TP, S , E 。对单功能或多功能、线性流水线、输入任务不连续的情况下,如何计算 TP, S , E 呢? 这只能通过画出正确的时空图,然后“数格子”求出答案。

【例 4-1】 如图 4-21 所示,有一条 4 段浮点数加法器流水线,求 $Z = A + B + C + D + E + F + G + H$ 浮点数之和。

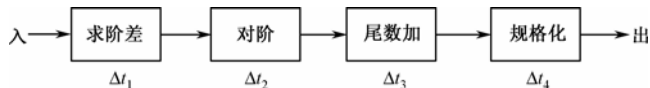


图 4-21 浮点数加法器流水线

解: $Z = A + B + C + D + E + F + G + H$ 。由于存在数据相关,依次相加,如同非流水线一样。现变形为

$$Z = [(A+B) + (C+D)] + [(E+F) + (G+H)]$$

则消除部分数据相关后可以连续输入到流水线中,其时空图如图 4-22 所示。设每个功能段延

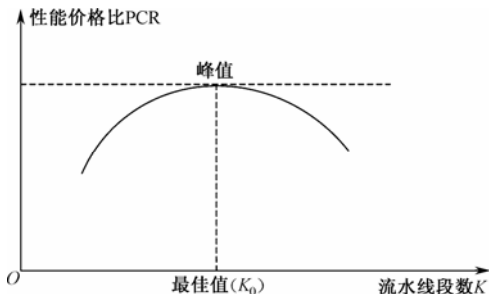


图 4-20 流水线的最佳段数

迟时间相等，均为 Δt_0 ，则 $T=15\Delta t_0$ ，输出中间结果和最后结果 $n=7$ ，吞吐率

$$TP = \frac{n}{T} = \frac{7}{15 \times \Delta t_0} = 0.47 \times \frac{1}{\Delta t_0}$$

加速比

$$S = \frac{T_{\text{非流水}}}{T_{\text{流水}}} = \frac{4 \times 7 \times \Delta t_0}{15 \times \Delta t_0} = 1.87$$

效率

$$E = \frac{\text{7 个任务占用的时空区}}{\text{4 个段总的时空区}} = \frac{4 \times 7 \times \Delta t_0}{4 \times 15 \times \Delta t_0} = 0.47$$

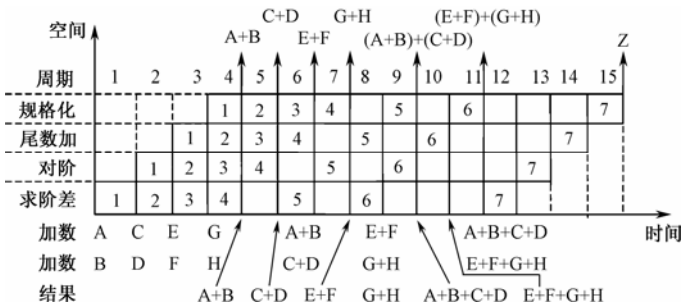


图 4-22 用一条 4 段浮点数加法器流水线求 8 个浮点数之和的流水线时空图

【例 4-2】 如图 4-22 所示，计算下式

$$Z=AB+CD+EF+GH$$

解：为了尽量减少数据相关性，充分发挥流水线作用，计算顺序应该先做 4 个乘法，然后做两个加法，最后求总的结果 Z，其时空图如图 4-23 所示。设每个功能段延迟时间相等，都是 Δt_0 ，则 $T=20 \times \Delta t_0$ ， $n=7$ 。吞吐率

$$TP = \frac{n}{T} = \frac{7}{20 \times \Delta t_0} = 0.35 \times \frac{1}{\Delta t_0}$$

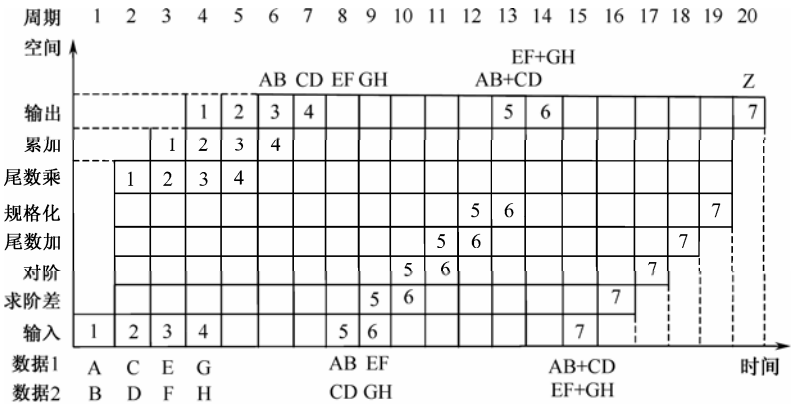


图 4-23 用 TI-ASC 多功能静态流水线求两个向量点积的流水线时空图

如果采用非流水线、顺序执行方式，一次乘法为 $4\Delta t_0$ ，一次加法为 $6\Delta t_0$ ，完成全部运算

$$T_{\text{非流水}}=4 \times 4 \times \Delta t_0 + 3 \times 6 \times \Delta t_0 = 34 \times \Delta t_0$$

则加速比

$$S = \frac{T_{\text{非流水}}}{T_{\text{流水}}} = \frac{34 \times \Delta t_0}{20 \times \Delta t_0} = 1.70$$

效率

$$E = \frac{7 \text{ 个任务占用的时空区}}{8 \text{ 个段总的时空区}} = \frac{34 \times \Delta t_0}{8 \times 20 \times \Delta t_0} = 0.21$$

整个流水线效率很低。原因如下：

- (1) 多功能流水线在做某种运算时，总有一些功能段空闲。
- (2) 静态流水线必须等前一种运算全部排出流水线后，才能启动下一种运算。
- (3) 本题有数据相关。
- (4) 流水线有装入与排空部分。

4.3 非线性流水线处理机

在线性流水线中，由于每个任务在流水线每一个功能段内只通过一次，因此可以在每个时钟周期（即每个节拍）向流水线输入一个新任务。这些任务不会争用同一个功能段，所以线性流水线调度非常简单。在非线性流水线中，由于存在反馈回路，当一个任务在流水线中流过时，在同一功能段可能要经过多次。因此就不能在每个节拍输入一个新任务，否则会发生在同一时刻有若干任务同时争用同一功能段的情况。这种情况称为功能部件冲突，或流水线冲突。

为了避免发生流水线冲突，一般采用延迟输入新任务的方法。由此就产生了非线性流水线的调度问题：间隔多少节拍向流水线输入一个新任务才能使流水线各个功能段都不发生冲突？在许多非线性流水线中，间隔的节拍数往往不是一个常数，而是一串周期变化的数字。因此，非线性流水线调度的目的是要找出一个最小的循环周期，按照这个周期向流水线输入新任务就不会发生流水线冲突，而且流水线的吞吐率和效率最高。

4.3.1 预约表和等待时间分析

非线性流水线的表示与线性流水线有两个明显的不同：

- (1) 非线性流水线有反馈线 and 前馈线。
- (2) 其输出端可以不在最后一个功能段，而可能从中间任意一个功能段输出。

1971年，E.S.Davidson 提出使用一个二维的预约表（Reservation Table）来描述一个任务（指令、操作）在非线性流水线中对各功能段的使用情况。预约表相当于时空图，预约表的行表示功能段，列表示时钟周期（即节拍， Δt 或 t ）。列数只反映非线性流水线处理一个任务的 Δt 数。所以预约表只反映流水线处理一个任务的时空流。静态非线性流水线的预约表呈现“对角线”形态，价值不大。而动态非线性流水线的预约表是分析的基础，不同功能组合可有不同的预约表，所以对同一个动态非线性流水线可以有多个预约表。静态非线性流水线每次启动用同一张预约表。动态非线性流水线对应不同功能、不同任务，使用不同预约表启动。预约表一行内有多个符号（ x 或 y ），表示不同 Δt 内重复使用同一段 S_i ；预约表一列内有多个符号（ x 或 y ），表示同一个 Δt 内不同的功能段并行工作。所以预约表可以分析动态非线性流水线对于多个性质相同的任务因为不恰当启动而引起的冲突，即如何保证不引发冲突而使动态非线性流水线的使用效率（吞吐率、加速比、设备效率）达到最佳。

图 4-24（a）所示是一条有三个功能段的多功能动态非线性流水线，除了从 S_1 到 S_2 和从 S_2 到 S_3 的流线连接，还有从 S_3 反馈到 S_2 、从 S_3 反馈到 S_1 的反馈线，有从 S_1 直接送 S_3 的前馈线。输入的任务有两个 x 和 y 。任务 x 输入，途径 $S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_2 \rightarrow S_3 \rightarrow S_1 \rightarrow S_3 \rightarrow S_1 \rightarrow$ 输出 x 结果。任务 y 输入，途径 $S_1 \rightarrow S_3 \rightarrow S_2 \rightarrow S_3 \rightarrow S_1 \rightarrow S_3 \rightarrow$ 输出 y 结果。由此，可得到任务 x 和任务 y 的两张预约表，如图 4-24（b），（c）所示。

流水线两次启动之间的时间间隔就是等待时间（Latency），用间隔的时钟周期数表示（即间隔的 Δt 数）。等待时间 k 是指两次启动之间有 k 个 Δt 的间隔，等待时间一定是正整数。

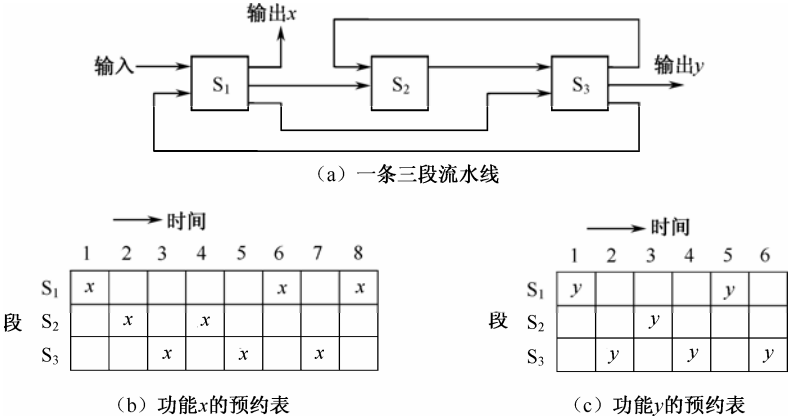


图 4-24 一条有两种不同功能并带有前馈和反馈连接的动态流水线

冲突（Collision）指在同一条流水线内，两次启动之间的资源冲突，即同一功能段在同一 Δt 内，出现不同的任务。引起冲突的等待时间称为禁止等待时间，简称禁止时间。不引起冲突的等待时间称为允许等待时间，简称允许时间。因此，在对流水线连续启动进行安排时，必须避免任何冲突，即必须避免在禁止时间内启动。以图 4-24（b）功能 x 的预约表为例。使用禁止时间为 2 引起的冲突如图 4-25（a）所示。使用禁止时间为 5 引起的冲突如图 4-25（b）所示。

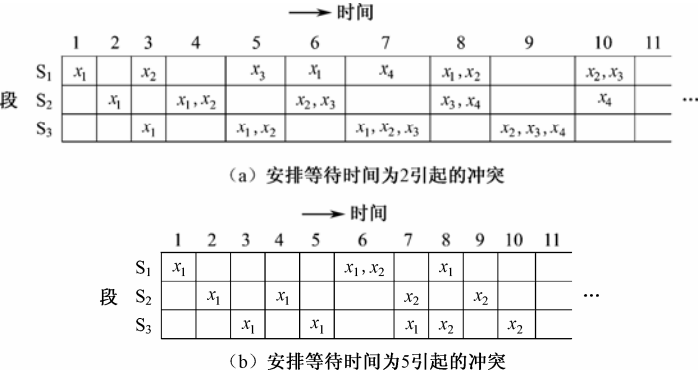


图 4-25 禁止等待时间为 2 和 5 所引起的冲突

使用允许时间就不会引起冲突，以图 4-24（b）功能 x 的预约表为例。使用等待时间为 1，8 循环（平均等待时间为 4.5）如图 4-26（a）所示；使用等待时间 3 循环（平均等待时间为 3）

如图 4-26 (b) 所示；使用等待时间为 6 循环（平均等待时间为 6）如图 4-26 (c) 所示。在允许时间内反复启动流水线也不会引起冲突，问题在于选择一个合适的允许时间，使流水线的使用效率较高。从图 4-26 可见，允许时间为 3 循环，流水线的效率较高。

测定禁止时间的方法如下：检查预约表内同一行中任意两个格子之间的距离（即间隔的格子数）即可。以图 4-24 预约表为例。功能 x 的预约表内， S_1 行内有 5 格（1→6）、7 格（1→8）、2 格（6→8）； S_2 行内有 2 格（2→4）； S_3 行内有 2 格（3→5）、4 格（3→7）、2 格（5→7）。归纳后， x 表的禁止时间为 2，4，5，7。功能 y 的预约表内， S_1 行有 4 格（1→5）； S_2 行内无； S_3 行内有 2 格（2→4）、4 格（2→6）、2 格（4→6）。归纳后， y 表的禁止时间为 2.4。

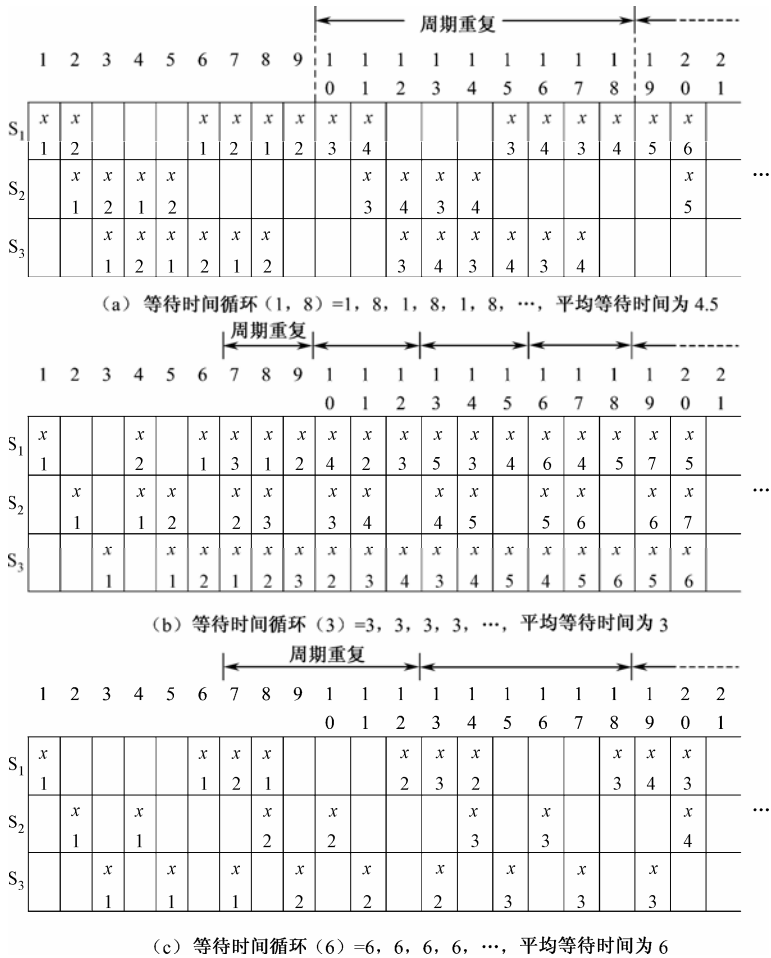


图 4-26 一条有两种不同功能并带有前馈和反馈连接的动态流水线

一个等待时间循环的平均等待时间由一次循环之内的所有等待时间之和除以等待时间种类数求得。例图 4-26 (a) 等待时间循环为 1，8，等待时间种类为 2，所以平均等待时间是 (1+8)/2=4.5。只有一个等待时间值的等待时间循环称为恒定循环，图 4-26 的 (b)，(c) 即是。恒定循环的平均等待时间就是它本身的等待时间。

4.3.2 无冲突调度

流水线调度的主要目的是使两次启动之间的平均等待时间最短，而不引起冲突。流水线无冲突调度的设计理论最早是由 Davidson 在 1971 年提出的，并与其学生一起逐步完善。它涉及冲突向量、状态图、简单循环、迫切循环和最小平均等待时间。

1. 冲突向量

对预约表进行分析，可以从禁止时间集合中识别出允许时间集合。对于一张 n 列的预约表，最大禁止时间 $m \leq n-1$ ，允许时间 p 在 $1 \leq p \leq m-1$ 范围内选择。 p 应尽可能小，理想的 $p=1$ ，即静态流水线的等待时间总是 1，可见于对角线或流线型预约表。所谓冲突向量(Collision Vectors)是指允许时间和禁止时间的集合。冲突向量是一个 m 位的二进制向量 $C=(C_m, C_{m-1}, \cdots, C_1)$ 。如果等待时间 i 引起冲突，则 $C_i=1$ ；如果等待时间允许启动（即进入流水线），则 $C_i=0$ 。注意： C_m 总等于 1，对应于最大等待时间（即冲突向量中最高位常等于 1）。例如，图 4-24 中 x 预约表， $m=n-1=8-1=7$ ，因为 x 预约表的禁止时间集合为 2, 4, 5, 7（见前述），则允许时间集合就是 1, 3, 6, x 预约表的冲突向量 C_x 为 7 位。从 $C_7 \sim C_1$ ，已知禁止时间集合为 2, 4, 5, 7，即 $C_7=C_5=C_4=C_2=1$ ，余下的 1, 3, 6 为允许时间，即 $C_6=C_3=C_1=0$ ，则 $C_x=1011010$ 。同理，图 4-24 中 y 预约表 $m=n-1=6-1=5$ ，因为 y 预约表的禁止时间集合为 2, 4（见前述），则允许时间集合就是 1, 3, 5。 y 预约表的冲突向量 C_y 按理应为 5 位，从 $C_5 \sim C_1$ ，已知禁止时间集合为 2, 4，即 $C_4=C_2=1$ ，余下的 1, 3, 5 为允许时间，即 $C_5=C_3=C_1=0$ ，则 C_y 应该为 01010，由于冲突向量规定最高位必须为 1，所以 $C_5=0$ 去掉， $C_y=1010$ 。

2. 状态图

根据上述冲突向量可以构造一张状态图(State Diagrams)，说明在相继的启动之间可允许状态的变换。流水线在起始时间（即第一格）时，初始状态相对应的冲突向量（上述的 C_x 或 C_y ）称为初始冲突向量。令 p 为在 $1 \leq p \leq m-1$ 范围内的可允许时间。建立一个 m 位的右移寄存器（如图 4-27 (a) 所示），实现逻辑右移运算。开始时寄存器内是初始冲突向量，此时， $p=0$ ，每右移一位， $p+1$ ，当移出为 1 时，表示冲突，此时的 p 为禁止时间；当移出为 0 时，表示安全，此时 p 为允许时间。每右移 p 位后，寄存器内状态与初始冲突向量进行逻辑加（即或运算）得到新状态进入寄存器。

以图 4-24 中 x 预约表为例。初始冲突向量 $C_x=1011010$ ，寄存器内初始为 1011010。

初始	1011010	$p=0$
右移 1 位	0101101	0 $p+1 \rightarrow p, p=1$ ，移出 0 安全
	<u>∨1011010</u>	
	1111111	寄存器内为 1111111
初始	1011010	
右移 2 位	0010110	10 $p+2 \rightarrow p, p=2$ ，移出最高位为 1，冲突
	<u>∨1011010</u>	
	1011110	寄存器内为 1011110
初始	1011010	
右移 3 位	0001011	010 $p+3 \rightarrow p, p=3$ ，移出最高位为 0，安全
	<u>∨ 1011010</u>	
	1011011	寄存器内为 1011011

对于图 4-24 中 y 预约表, $C_y=1010$, 有

初始	1010	$p=0$
右移 1 位	0101	0 $p+1 \rightarrow p, p=1$, 移出 0, 安全
$\vee 1010$		
	1111	寄存器内为 1111
初始	1010	
右移 2 位	0010	10 $p+2 \rightarrow p, p=2$, 移出最高位为 1, 冲突
$\vee 1010$		
	1010	寄存器内为 1010
初始	1010	
右移 3 位	0001	010 $p+3 \rightarrow p, p=3$, 移出最高位为 0, 安全
$\vee 1010$		
	1011	寄存器内为 1011

在图 4-27 (b) 中, 得到了功能 x 的状态图。从初始状态 (1011010) 只可能有三个输出变换, 对应于三个可允许时间 6, 3, 1。类似地, 从状态 (1011011) 右移 3 位或右移 6 位后, 就达到同一状态。当移位次数 $p \geq m+1$ 时, 寄存器内返回到初始状态。例如, 右移 8 位或更多位 (记作 8^+) 后, 下一个状态一定是初始状态。同理, 在图 4-27 (c) 中得到了功能 y 的状态图。状态图要点如下:

- (1) 只画出了允许时间的状态变化。
- (2) $\xrightarrow{3}$ 表示从初始状态转到结束状态, 其上数字表示右移位数。
- (3) $\xrightarrow{3}$ 表示从此状态再右移 3 位恢复到原状态。
- (4) 8^+ 中表示大于等于 8 的右移位数。
- (5) 1^* 、 3^* 中 $*$ 表示迫切循环 (见下述)。

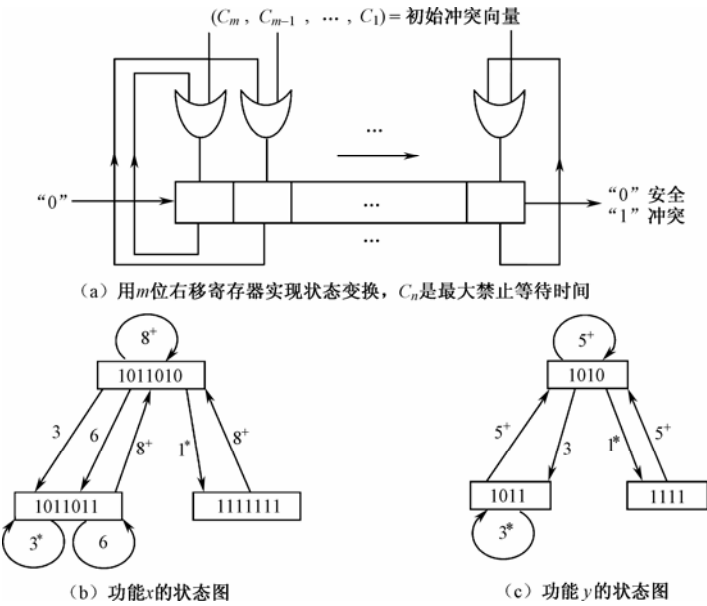


图 4-27 从图 4-24 两张预约表分别得到状态图

3. 迫切循环

寻找最小平均等待时间（Minimal Average Latency, MAL）是流水线调度中需解决的问题。从状态图中可确定形成 MAL 最佳等待时间循环。在状态图内可组合出多种允许时间的循环。例如图 4-27（b）中可得到（1，8），（1，8，6，8），（3），（6），（3，6），（3，8），（3，6，3）…多种循环组合。从中可找到简单循环——每个状态只出现一次的等待时间循环。如（3），（6），（8），（1，8），（3，8），（6，8）为简单循环，而（1，8，6，8）就不是简单循环，因为二次穿越（1011010）状态。类似地，（3，6，3，8，6）也不是简单循环，因为它三次重复状态（1011011）。

简单循环中有迫切循环，所谓迫切循环（Greedy Cycles）是指从各自初始状态输出的边缘都具有最小等待时间，即平均等待时间比其他等待时间更小。例如图 4-27（b）中，（1，8）和（3）是迫切循环；图 4-27（c）中，（1，5）和（3）是迫切循环。在功能 x 表中，（1，8）的平均等待时间为 $(1+8)/2=4.5$ ，（3）的平均等待时间恒定为 3，比简单循环（6，8）的平均等待时间 $(6+8)/2=7$ 更小。因此，构成迫切循环的条件是：首先必须是简单循环；其次，它的平均等待时间比其他简单循环更小。所谓寻找 MAL 就是在简单循环中的迫切循环内找到平均等待时间最小者。例如图 4-27（b）中，迫切循环用*号标记，而 MAL=3。

图 4-27（c）中，MAL=3，所以从若干迫切循环中可找出 MAL。因此流水线无冲突调度可以归纳为从简单循环集合中找出迫切循环，然后选定产生 MAL 的那个迫切循环。

4.3.3 流水线调度优化

基于 MAL 的流水线调度优化技术的基本思想是：将非计算延迟段（不占计算资源的延迟功能段）插入原来的流水线，然后修改预约表，产生新的冲突向量和改进的状态图，得到最佳的、最短的 MAL。

1. MAL 的限制范围

1972 年，Shar 提出了 MAL 限制范围三原则。其前提是：静态可重构流水线上，用任何控制策略执行给定预约表可以得到的范围。三原则如下：

- （1）MAL 的下限是预约表任一行中格子内符号的最大个数。
- （2）MAL 小于等于状态图中任一迫切循环的平均等待时间。
- （3）MAL 的上限是初始冲突向量中 1 的个数再加 1，也是任何迫切循环平均等待时间的上限。

例如，图 4-24（b）中，功能 x 的预约表内， S_1 行和 S_3 行均有 3 个 x ，所以 MAL 下限为 3。图 4-27（b）中，功能 x 的冲突向量是 1011010，1 的个数是 4，MAL 上限为 $4+1=5$ 。

又如，图 4-24（c）中，功能 y 的预约表内， S_3 行有 3 个 y ，所以 MAL 下限为 3。图 4-27（c）中，功能 y 的冲突向量是 1010，1 的个数是 2，MAL 上限为 $2+1=3$ 。

优化 MAL 的方法是设法修改预约表，使任一行格子内符号的最大数目减小，即使 MAL 的下限减小。但修改后的预约表必须保持原来的功能。Patal 和 Davidson 在 1976 年提出用插入非计算延迟段来提高流水线的性能。

2. 插入延迟

插入延迟是要修改预约表，以形成一个新的冲突向量，从而得到一张修改后的状态图，产生具有 MAL 下限的迫切循环。

图 4-28 (a) 所示为一个三段流水线，其预约表为图 4-28 (b)，得到的初始冲突向量为 $C_x=1011$ ，相应的禁止时间为 1, 2 和 4，得到的状态图如图 4-28 (c) 所示，迫切循环为 (3)，它的 $MAL=3$ 。但从预约表分析，任一行中格子内最大符号个数为 2，即 MAL 的下限为 2，因此图 4-28 (c) 得到的 $MAL=3$ 并不是最佳的。

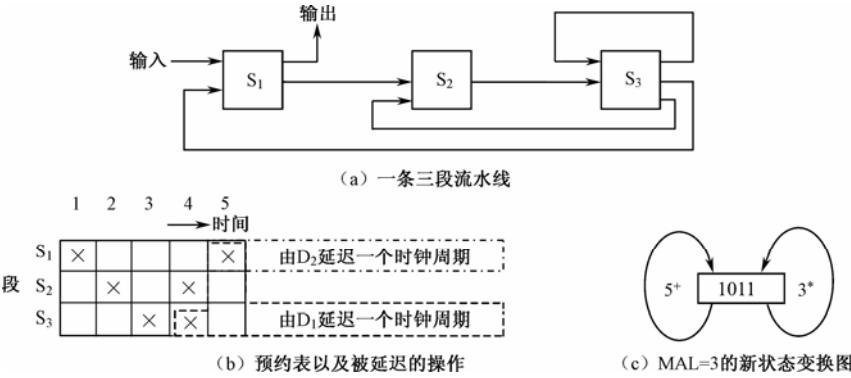


图 4-28 一条最小平均等待时间为 3 的流水线

将图 4-28 (a) 的流水线进行改造，在 S_3 段之后插入非计算段 D_1 , D_2 ，如图 4-29 (a) 所示。 D_1 的作用是将 x_1 从时间 4 延迟一个周期。相应地使 x_2 从原来的时间 5 也延迟一个周期。在第二次使用 S_1 之前再插入 D_2 ，使 x_2 从时间 6 再延迟一个周期。构成的预约表如图 4-29 (b) 所示，此时的预约表有 $3+2=5$ 行及 $5+2=7$ 列。总之， x_1 从时间 4 到时间 5 延迟了一个周期， x_2 从时间 5 到时间 7 延迟了两个周期，其余操作没有改变。从新的预约表得到新的初始冲突向量 $C_x=100010$ 以及修改后的状态图 (图 4-29 (c))。从状态图可得到简单循环 (1, 3), (3, 5), (1, 7), (3, 4), (3, 7), (5), (4, 7), ..., 其中迫切循环为 (1, 3)，产生的 $MAL=(1+3)/2=2$ ，比原来减小了，达到了 MAL 的下限，提高了流水线性能。

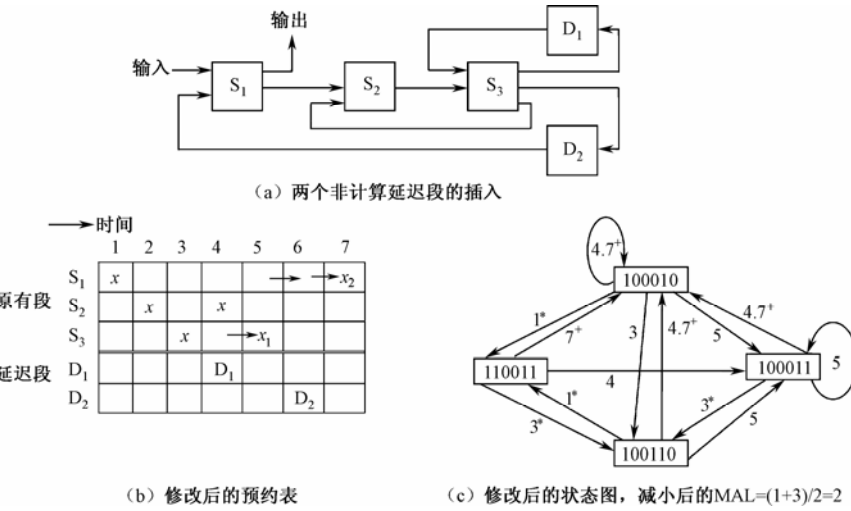


图 4-29 在图 4-28 的流水线中插入两个延迟段以获得最佳 MAL

3. 流水线的吞吐率

流水线吞吐率实质上就是启动速率，或者是每个时钟周期启动任务的平均数。如果在 n 个 Δt 内启动了 N 个任务，则吞吐率可用 N/n 计算。吞吐率也可用 MAL 的倒数计算。因此，调度优劣会影响流水线的性能。 MAL 越小，则吞吐率越高。由于 $1 \leq MAL \leq$ （任何迫切循环的最短等待时间），因此 $MAL=1$ 时，吞吐率最高，即每个 Δt 启动一个任务。上例功能 x 的 $MAL=3$ （图 4-27（b）），则吞吐率

$$TP = \frac{1}{MAL \square \Delta t} = \frac{1}{3 \Delta t}$$

若 $\Delta t=20ns$ ，则

$$TP = \frac{1}{3 \times 20 \times 10^{-9}} = 16.7MIPS$$

4. 流水线的效率

流水线的每个段流过足够长的一串任务，从而得到时间利用的百分数，就是该段的利用率。所有段的利用率累加起来就是流水线的效率。图 4-26（b）中等待时间循环（3），则效率 $E = \frac{8}{3 \times 3} = 88.8\%$ 。图 4-26（a）中等待时间循环为（1，8），则效率 $E = \frac{16}{3 \times 9} = 59.3\%$ 。图 4-26（c）中等待时间循环为（6），则效率 $E = \frac{8}{3 \times 6} = 44.4\%$ 。在图 4-26（a），（c）中的周期内，没有一个段是全利用的，而图 4-26（b）中，在周期内 S_1 和 S_3 被全利用，所以效率 E 最高。

流水线的吞吐率和效率是彼此相关的，吞吐率高是由等待时间循环短引起的，效率高说明流水线段的空闲时间少，利用率高。吞吐率和效率之间的关系是预约表的函数，也是相应的启动循环的函数。在任何可以接受的启动循环中，进入稳定状态下（即进入周期重复状态下），流水线中至少有一个段利用率为 100%。否则，流水线的能力就没有全部发挥出来。此时就需要进行流水线调度优化。

4.4 流水线相关处理

要使流水线发挥高效率，就要使流水线连续不断地流动，尽量不出现“断流”情况。但是“断流”现象还是出现了，其原因除了编译形成的目标程序不能发挥流水结构的作用，或存储系统供不上为连续流动所需的指令和操作数以外，就是出现了相关、转移和中断问题。

因为转移、中断等现象与它们之后的指令有关联，从而不能同时解释，我们称为全局性相关。那些与主存操作数或寄存器的“先写后读”等关联，我们又称为局部性相关。

4.4.1 局部相关及处理

在实际的流水线结构里要解决可能出现的相关问题，按流水的流动顺序的安排与控制，可有下列几种。

1. 顺序流动

顺序流动是使流水线输出端的任务（指令）流出顺序与输入端的流入顺序一样，遇到相关，则使相关的指令解释过程停止，待过了相关这一环节再继续下去。

假如有一串指令的执行顺序是先执行 h ，然后依次是执行 i, j, k, \dots ，它们将在有 8 个

功能段的流水线里流动，如图 4-30 所示。一般流水线里读段在前，写段在后。如果有 h 和 j 两条指令对同一存储单元有“先写后读”的要求，当第 j 条指令到达读段时，第 h 条指令还没到达写段，则第 j 条指令读出的数是错的，这就是“先写后读”相关。解决的方法是，当第 j 条指令到达读段时发现与第 h 条指令相关，则第 j 条指令在读段停止，当第 h 条指令流过写段后，第 j 条指令才继续流下去。这样解决了相关，但出现了空段，当然会降低流水线的效率和吞吐率，这种方法控制起来比较简单。

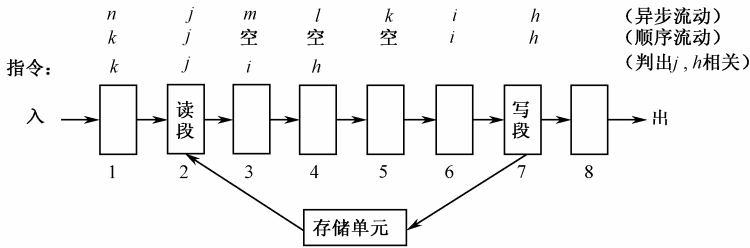


图 4-30 顺序流动和异步流动

2. 异步流动

异步流动是使流水线输出端的任务（指令）流出顺序与输入端的流入顺序不一样，这是鉴于顺序流动停止时， j 指令以后的指令串都停止，以期保持流入与流出顺序一样。但如果 j 指令以后的指令与已进入流水线的所有指令都没有相关问题，那么完全可以越过 j 指令（ j 指令仍在读段停止）进入流水线向前流动，这样处理就使流入与流出的顺序不一样了，称为异步流动。

由于实际流水线中各段对不同指令执行时间可能不一样，且有些段对某些指令不必执行，而允许超越前面指令而流动的异步流动流水线还会出现“写-写”相关和“先读后写”相关。例如，第 k , j 条指令都有写操作，且写入同一存储单元，若出现第 k 条指令先于第 j 条指令到达写段，则该存储单元内容最后是由第 j 条指令写入的而非第 k 条指令写入的，这种情况就是“写-写”相关。又例如，第 j 条指令的读操作和第 k 条指令的写操作是同一存储单元，若出现第 k 条指令的写操作先于第 j 条指令的读操作，则第 j 条指令读出的是位于其后第 k 条指令写入的内容，由此产生错误，这种情况就是“先读后写”相关。这两种相关仅出现在异步流动流水线中，显然采用异步流动流水线能提高整个流水线的效率和吞吐率，但要解决超越和新出现的另两种相关，会使控制变得复杂。

3. 相关专用通路

在上面的例子里，第 h 条指令与第 j 条指令发生“先写后读”相关时，由于第 j 条指令读操作要从存储单元读出应该由第 h 条指令写入存储单元的内容，但第 h 条指令还来不及写入，我们设想在读段与写段有一个捷径——专用通路，第 j 条指令不从存储单元读，而是从第 h 条指令在写段的数据来读，就可以避免第 j 条指令读操作的停止，这就是相关专用通路的概念。

当然，可以采用停止的方法以降低速度来达到解决相关的目的，而专用通路是以增加设备来解决相关问题。

4.4.2 全局相关及处理

相关转移是全局性相关。当出现条件转移指令时，为了保证流水线能够继续向前流动，主要采用猜测法，它的基础是条件转移指令是否实现转移，其前提是转移条件得到满足。因此为了保持流水的顺序，我们采用不满足转移猜测，即流水线在遇到条件转移指令时，猜它不满足转移条件，而继续解释它以后的指令，一旦在其后的“判断条件段”发现满足转移条件时，再转移，这样仅仅报废几条指令的解释，如图 4-31 所示。

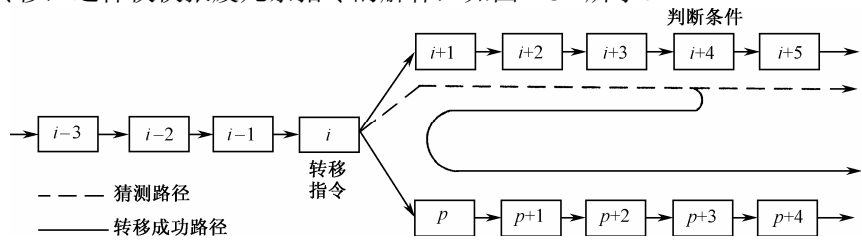


图 4-31 用猜测法处理条件转移相关

全局相关的猜测法设计的关键是应保证在猜错而需返回到分支点时，能恢复分支点处的原有现场。

4.4.3 流水线中断处理

中断往往不能预知。由于流水线里流动着好几条指令，当发生中断时，如何使该中断指令的现场和其后已进入流水线的指令得到保护，以及如何恢复中断是设计的关键。

一种简便的方法是利用“不精确断点法”，不论第 i 条指令在流水线的哪一段发出中断申请，那时还没进入流水线的后继指令就不再允许进入，但已在流水线的指令仍然流动直到执行完毕，而后才转入中断处理程序。不精确断点是指处理机并非中断在第 i 条指令处，只有当第 i 条指令在第一段时发生中断才是精确断点。

这种“不精确断点法”对程序设计者很不方便，程序调试时尤其如此，后来多采用“精确断点法”。不论第 i 条指令在流水线中的哪一段发中断请求，中断处理的现场都是对应第 i 条指令的，而且在第 i 条指令后已进入流水线的指令的现场都能恢复。显然，“精确断点法”需采用很多后援寄存器，以保证流水线内的各条指令的原有状态都能保存和恢复。

4.5 向量的流水处理和向量处理机

4.5.1 向量处理基本概念

从标量流水线处理机分析中可知，如果输入流水线的指令既无局部性相关，也无全局性相关，则流水线可能装满。此时，可获得高吞吐率和效率。在科学计算中，往往有大量不相关的数据进行同一种运算，这正适合于流水线特点。因此就出现了设置有向量数据表示和向量指令的向量流水线处理机。由于这种机器能较好地发挥流水线技术特性，因此可达到较高速度，一般称向量流水处理机为向量机（Vector Processor）。

下面用一个简单的例子说明向量的流水处理，用循环程序可以完成向量运算：

DO 10 $i=1, n$

10 $d_i=a_i*(b_i+c_i)$

对此，可以有下面几种方法。

1. 水平（横向）处理法

如果用逐个求 d_i 的方式，则

$$d_1=a_1*(b_1+c_1)$$

$$d_2=a_2*(b_2+c_2)$$

...

$$d_i=a_i*(b_i+c_i)$$

...

$$d_n=a_n*(b_n+c_n)$$

这种方法中每次循环至少要用两条指令：

$$k_i=b_i+c_i$$

$$d_i=k_i*a_i$$

显然在流水处理中，不仅有操作数相关（“先读后写”相关），而且每次循环中又有功能的切换（+，*，+，...），这就使流水线的效率和吞吐率降低。

2. 垂直（纵向）处理法

实际上，可以认为 A，B，C，D 是长度为 n 的向量。

$$A=(a_1, a_2, \dots, a_n)$$

$$B=(b_1, b_2, \dots, b_n)$$

$$C=(c_1, c_2, \dots, c_n)$$

$$D=(d_1, d_2, \dots, d_n)$$

因此，上述 DO 循环语句可写成下列向量运算形式

$$D=A*(B+C)$$

改进的办法是对整个向量按相同的运算处理完之后，再去执行别的运算，即

$$B+C \rightarrow K(1 \sim n)$$

$$K*A \rightarrow D(1 \sim n)$$

仅用两条向量指令，且处理过程中没有出现转移，每条向量指令内无相关，两条向量指令间只有一次相关。而且只有一次功能切换。这种处理方法就是向量的流水处理，从流水线输出端可以每拍取得一个结果元素。

由于向量长度 n 是不受限制的，无论 n 有多大，相同运算都用一条向量指令完成，因此，向量指令的源向量和目的向量都在主存内，流水线运算部件的输入、输出端通过缓冲器与主存连接，构成“存储器-存储器”型的运算流水线。如图 4-32 所示，这种结构要求提高主存和流水线处理机之间的信息通信流量，使流水线

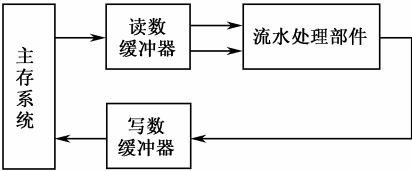


图 4-32 “存储器-存储器”型的运算流水线

输入每个节拍从主存取到元素并向主存写回一个结果，这样才能保证流水线的平稳流动。

3. 分组（纵横）处理法

把长度为 N 的向量分成若干组，每组长度为 n ，组内按纵向方式处理，依次处理各组，若

$$N=sn+\gamma$$

式中， γ 为余数，也作为一组处理，则共有 $s+1$ 组，若

第一组

$$K_{1\sim n}=B_{1\sim n}+C_{1\sim n}$$
$$D_{1\sim n}=K_{1\sim n}*A_{1\sim n}$$

第二组

$$K_{(n+1)\sim 2n}=B_{(n+1)\sim 2n}+C_{(n+1)\sim 2n}$$
$$D_{(n+1)\sim 2n}=K_{(n+1)\sim 2n}*A_{(n+1)\sim 2n}$$
$$\vdots$$

第 $s+1$ 组

$$K_{(sn+1)\sim N}=B_{(sn+1)\sim N}+C_{(sn+1)\sim N}$$
$$D_{(sn+1)\sim N}=K_{(sn+1)\sim N}*A_{(sn+1)\sim N}$$

每组内各用两条向量指令，仅有一次向量指令相关，因此适合于对向量流水处理，由于向量总长度 N 不受限制，而组内不准超越 n 。因此，可设长度为 n 的向量寄存器，使每组向量运算的源向量和目的向量均在向量寄存器中，流水线的输入、输出端与向量寄存器相连，构成“寄存器-寄存器”型运算流水线，如图 4-33 所示。

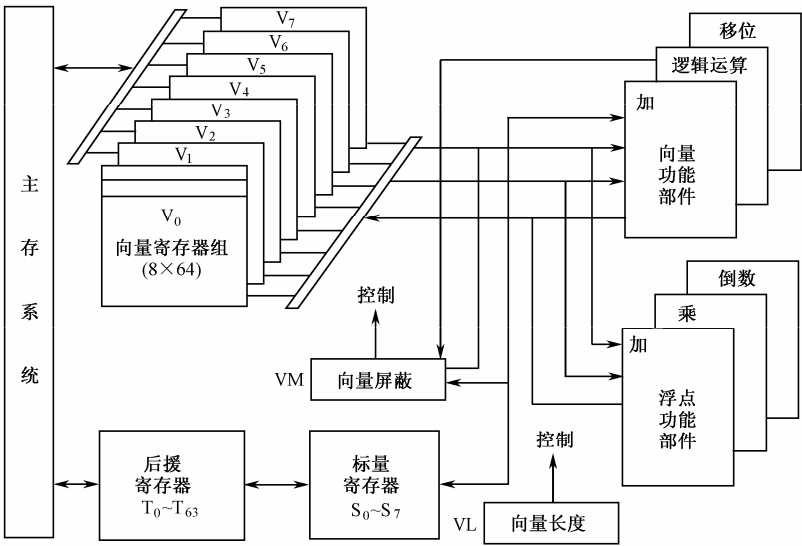


图 4-33 “寄存器-寄存器”型运算流水线

CRAY-1 型巨型机、国产银河-1 型巨型机的 CPU 均采用这种结构，它有容量相当大的向量寄存器组，可保存源向量和中间结果，大大减少访存次数，降低对存储器信息通信流量的要求，减少因存储器冲突而等待的时间，大大提高了处理速度。

标量处理机执行一条运算指令可得到一个结果，因此通常用每秒执行百万条指令 MIPS (Million Instructions Per Second) 衡量机器速度。而向量处理机一条指令可得到几十个甚至更多的运算结果，用 MIPS 衡量不合适，因此用每秒取得百万个浮点运算结果来表示，以 MFLOPS (Million FLOating-point instnution Per Second) 为测量单位。MIPS 把服务性指令都

计算在内，而 MFLOPS 不考虑服务性指令，故两者不能等同。一般认为，标量计算机执行一次浮点运算需 2~5 条指令，平均需 3 条指令，所以 $MIPS=3MFLOPS$ 。

向量流水处理有下列主要特点：

- (1) 在向量操作中，每个向量元素的计算与其他向量元素的计算是相互独立的，即向量元素的运算不存在数据相关，从而使向量流水线有较深的深度，也清除了数组标量运算中循环控制开销。
- (2) 一条向量指令中有大量数据运算，相当于一个标量循环，从而降低对指令访存带宽的要求。
- (3) 向量元素在存储器中存放地址均相邻，即是连续的，可采用低位交叉存储方式提高访问速度。

上述特点使得对相同数量的数据进行操作时，向量操作比一串标量指令操作更快。同时，向量流水处理可使访存和寻址方式中的有效地址计算流水化，通过设置多个向量运算部件，允许多个向量操作同时进行，从而开发对不同元素进行多向量操作的并行性。

4.5.2 向量处理机的结构

向量处理机系统结构分为“存储器-存储器”型（见图 4-32）和“寄存器-寄存器”型（见图 4-33）两大类。

“存储器-存储器”型向量处理机中，主存由多个模块构成，流水处理部件与主存系统之间有三条独立的数据通路（两条输入，一条输出），各数据通路可以同时工作，但一个存储模块在某一时刻只能为一个通路服务。其工作特点是，源向量都取自主存，且结果向量也存放 to 主存中。

“寄存器-寄存器”型向量处理机中，主存系统和向量功能部件之间插入了一个小容量的高速向量寄存器组，可得到较大的带宽。为了最大限度地利用高速向量寄存器组，应尽量使大部分操作在向量寄存器组之间进行，减少访存次数，降低对主存带宽的要求。其工作特点是，源向量都取自向量寄存器，且结果向量也存放到向量寄存器中。

典型的向量处理机的基本系统结构如图 4-34 所示，它由一个标量流水部件和若干向量流水部件组成。基本结构包括标量寄存器、标量功能部件、向量存取部件、向量寄存器或向量缓冲部件、向量功能部件以及向量控制部件等。

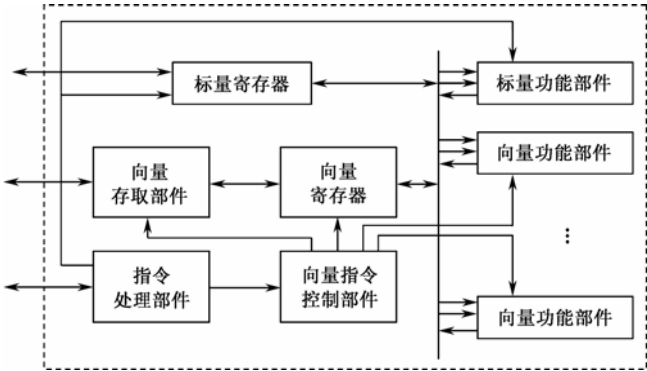


图 4-34 向量处理机的基本系统结构

早期的向量机都是“存储器-存储器”型，如 TI 公司的 ASC（1972 年），CDC 公司的 STAR-100（1973 年）、CYBER-205（1982 年）和 ETA-10（1986 年）。1976 年 CRAY 公司研制的 CRAY-1 向量机首先采用“寄存器-寄存器”型，由于它在短向量操作时有良好的性能，指令系统简洁，因此“寄存器-寄存器”型逐步成为向量机的主流。美国 Cray 公司的 CrayY-MP（1988 年）和 C-90（1991 年），日本 Fujitsu 公司的 VP2000（1991 年）和 VPP300/500（1993 年）等大规模超级向量流水线处理机均属于这种类型。

在基于“寄存器-寄存器”型的向量机中，向量指令有下列 6 种类型：

（1）向量-向量指令（Vector-Vector Instruction）。从向量寄存器中取一个或两个向量操作数，送入向量流水线进行相应的运算，结果放入另一个向量寄存器。

（2）向量-标量指令（Vector-Scalar Instruction）。参加运算的一个操作数来自向量寄存器，另一个操作数来自标量寄存器，标量数据与向量数据每个元素进行运算。

（3）向量-存储器指令（Vector-Memory Instruction）。在向量寄存器和存储器之间进行向量数据传送。

（4）向量归约指令（Vector Reduction Instruction）。操作数来自向量寄存器，操作结果为标量数据，如从一个向量中找出最大值、最小值和中间值等。

（5）聚集-散射指令（Gather and Scatter Instruction）。用两个向量寄存器分别存放数据和变址值。聚集指令根据变址值，并把存储器中某个稀疏向量的非零元素取出放到向量寄存器内。散射指令进行逆操作，把一个向量以稀疏向量的形式存入存储器，其非零项由变址值指出。

（6）屏蔽指令（Masking Instruction）。利用屏蔽向量把一个向量压缩或展开成一个较短或较长的索引向量。

CRAY-1 是一种典型的向量处理机，该系统由一台前置机（标量机）和后台的 CRAY-1 向量机组成，前置机起系统管理的作用。CRAY-1 向量机有 120 种指令，10 种格式的向量，13 种格式的标量。主振 80MHz（时钟周期 12.5ns），运算速度在 20~80MIPS 之间，最大的浮点运算速度为 160 MFLOPS。CRAY-1 向量处理机结构框图如图 4-35 所示。

1. 寄存器组

CRAY-1 向量处理机的寄存器组一共有 5 种：

（1）向量寄存器组（V）。由 512 个 64 位的寄存器组成，分成 8 组（ $V_0 \sim V_7$ ），每组可存放最多包含 64 个分量的一个向量，所以向量寄存器中最多可存放 8 个向量。向量的一组数据可以存放在一组向量寄存器中的若干分量寄存器中。向量操作数的长度放在向量长度寄存器 VL 内，VL 长 6 位。

（2）标量寄存器组（S）。由 8 个 64 位寄存器（ $S_0 \sim S_7$ ）组成，用于存放执行标量的算术和逻辑运算指令时的操作数。

（3）地址寄存器组（A）。由 8 个 24 位寄存器（ $A_0 \sim A_7$ ）组成，用于存放主存地址、变址值、移位计数值、循环计数值和 I/O 通道地址等。

（4）标量缓冲寄存器组（T）。由 64 个 64 位的标量缓冲寄存器组成。S 寄存器组可直接访问主存，也可经过 T 寄存器组传送数据。

（5）地址缓冲寄存器组（B）。由 64 个 24 位的地址缓冲寄存器组成，存放在一段时间内反复使用的地址。A 寄存器组可以直接访问主存，也可通过 B 寄存器组传送地址。

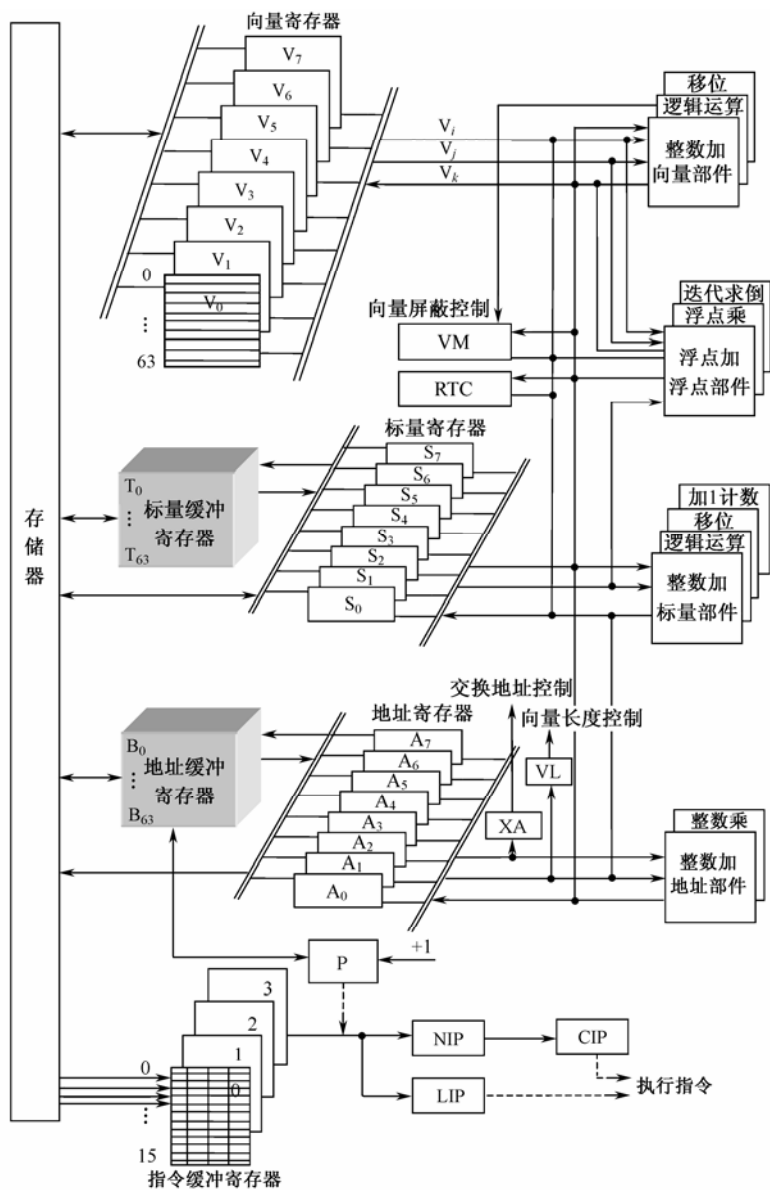


图 4-35 CRAY-1 向量处理机结构框图

T 和 B 是中间寄存器，只是分别对 S 和 A 提供支持。P 寄存器是指令计数器，存放当前指令地址。图 4-35 左下角有 4 组指令缓冲寄存器。每组有 64 个 16 位的寄存器，指令长度有 16 位和 32 位两种。指令缓冲寄存器组可实现指令预取技术。NIP，CIP，LIP 寄存器用来发送指令。VM 为 64 位的向量屏蔽寄存器，可在运算前屏蔽向量寄存器中某些分量。XA 为交换地址寄存器，RTC 为实时时钟计数器。

2. 多个单功能流水部件

CRAY-1 向量机有 12 个独立的单功能流水部件。三个用于向量运算：整数加、逻辑运算、移位；三个用于浮点运算：浮点加、浮点乘、浮点迭代求倒数；4 个用于标量运算：整数加、逻辑运算、移位、加 1 运算；两个用于地址运算：整数加、整数乘。这些功能部件都采用流

水线结构。各功能部件的流水经过的时钟是不一样的，但满负荷流动时每个时钟流出一个结果。为了充分发挥向量寄存器组与向量功能部件和浮点功能部件的作用以及加快向量处理， $V_0 \sim V_7$ 中任一组向量寄存器都有单独的输入总线和单独的输出总线连接到 6 个功能部件。图 4-35 中的 V_i, V_j, V_k 表示的不是公共总线，而是若干组单独总线。只要不发生向量寄存器冲突和功能部件冲突，各组向量寄存器之间和各功能部件之间都能并行工作，这就大大加快了指令处理速度。

3. 运算流水线

向量、标量和地址运算都采用运算流水线。CRAY-1 的 4 种典型向量指令如图 4-36 所示。第 1 种是“向量-向量”运算： V_i 和 V_j 向量分量流入 n 个时钟周期运算流水线，结果进入 V_k ；第 2 种是“向量-标量”运算： V_i 向量分量与 S_j 标量进入运算流水线，结果进入 V_k ；第 3 和第 4 种是在“主存-向量寄存器”组之间传送数据，传送的延迟时间均为 6 个时钟周期。运算流水线一旦满负荷，每个时钟周期送出一个结果向量分量。分量的个数由 VL 寄存器控制。在向量合并或测试时，由 VM 寄存器控制合并和测试的对象（即分量）。一条向量指令最多只能处理 64 对分量。如果向量的长度 $N > 64$ ，向量寄存器放不下，则必须执行向量循环，每条向量指令处理 64 个分量（即元素），循环 $\lceil N/64 \rceil$ 次。

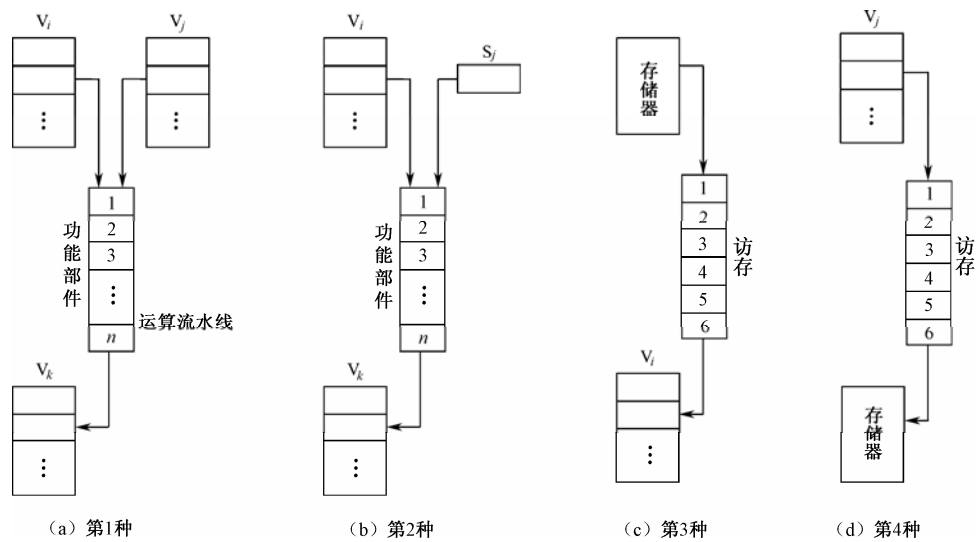


图 4-36 CRAY-1 向量处理机的 4 种向量指令

4.5.3 提高向量处理机性能的方法

研制高性能向量处理机要解决下列几个主要问题：

(1) 较好地维持向量、标量性能平衡。向量机具有处理向量和标量的功能，但处理这两类计算必须平衡。向量平衡点（Vector Balance Point）是指为使向量硬件设备和标量硬件设备利用率相等，在程序中向量代码所占百分比。其目标是花在向量硬件和标量硬件上的时间相等，系统资源不会空闲。如某系统向量运算速度达到 9MFLOPS，标量运算速度达到 1MFLOPS，程序代码 90% 是向量计算，10% 是标量计算，此时两种模式上的计算时间相等，那么该系统向量平衡点为 0.9。系统设计时应该保持足够高的向量平衡点，与用户程序向量化

程度相匹配。通常每台处理机重复设置流水线功能部件，或者向量部件采用超流水线技术，都可以提高向量运算性能。

(2) 可扩展性随着处理机数目的增加而提高。可扩展性是指在确定的应用背景下向量机系统性能随处理机数增加而线性提高。可扩展性的三个目标是：① 规模可扩展性，即系统设计的资源部件个数从小到大可扩展；② 换代可扩展性，即系统使用的处理器芯片和存储器芯片是可升级换代的，而且要求与换代后的新硬件兼容，可移植的软件及算法也应该是可扩展的；③ 问题可扩展性，问题是指数据集，问题可扩展的计算机在问题规模增大时仍能高效运行。

(3) 增加存储器系统的容量和性能。大规模存储器系统必须为标量处理提供低时延，为向量处理提供高频宽，为解决大型复杂问题提供大容量和高吞吐率的性能，存储器必须采用高效的层次结构。

(4) 提供高性能的 I/O 和易访问的网络。由于向量机每升级换代一次，其速度至少增加 3~5 倍，能处理的问题规模也增加了，对 I/O 频宽要求也要提高。向量机对高速连网能力的支持将变成 I/O 系统结构的主要组成部分。

提高向量处理性能的常用技术有多功能部件并行操作，链接技术，条件语句和稀疏矩阵加速处理，向量归约操作的加速处理。前两种技术是为了提高相邻的两条或多条向量指令的执行速度，后两种技术是为了使循环向量化，提高向量处理速度。

1. 多功能部件并行操作

向量寄存器冲突是指并行工作的各向量指令的源向量或结果向量使用了相同的向量寄存器。它有下列 5 种情况

(1) 源向量冲突

$$V_1 + V_2 \rightarrow V_3$$

$$V_1 * V_4 \rightarrow V_5$$

此例中两条向量指令均使用 V_1 作为源向量，由于两条向量指令使用的 V_1 的第一个元素（即首下标）可能不同，向量长度也可能不同。因此， V_1 不能同时向两条向量指令提供所需要的源向量，故不能并行执行。

(2) 结果向量冲突

$$V_1 + V_2 \rightarrow V_3$$

$$V_4 - V_5 \rightarrow V_3$$

此例中两条向量指令都使用 V_3 作为结果向量，存在冲突。

(3) “先读后写”的向量冲突

$$V_1 + V_2 \rightarrow V_3$$

$$V_4 * V_5 \rightarrow V_1$$

此例中第一条向量指令使用 V_1 作为源向量，第二条向量指令使用 V_1 作为结果向量，存在“先读后写”冲突。

(4) 源-目的向量相关

$$V_1 + V_2 \rightarrow V_3$$

$$V_3 * V_4 \rightarrow V_5$$

此例中 V_3 存在“先写后读”的源-目的向量相关。由于在向量机中，任何一个向量寄存器 V_i 在同一时钟周期内可以接收一个结果分量，并能为下一次操作再提供一个源分量，因此

存在源-目的向量相关的两条向量指令在不发生其他 V_i 冲突和功能部件冲突的前提下，可通过连接机构将两条向量指令的处理过程连接起来，实现两条指令的流水处理。

(5) 功能部件冲突

$$V_1 * V_2 \rightarrow V_3$$

$$V_4 * V_5 \rightarrow V_6$$

此例中两条向量指令均使用了相同的乘法功能部件，因此存在功能部件冲突。

上述的冲突和相关使两条向量指令不能并行操作。由于向量机可以具有多个单功能流水部件和运算流水线，并且都有单独的输入/输出数据总线，因此只要不发生向量寄存器冲突和功能部件冲突，就可以并行处理，同时实现两条向量指令的流水处理，如

$$V_1 + V_2 \rightarrow V_3$$

$$V_4 * V_5 \rightarrow V_6$$

如果若干向量指令均符合上述原则，均可并行执行，其前提是有足够多的向量寄存器和功能部件及运算流水线。

2. 链接技术

链接技术是流水线中加快运算速度的一种重要技术。在不出现源向量冲突、结果向量冲突、“先读后写”向量冲突和功能部件冲突时，通过链接部件将有源-目的向量相关的前后两条或多条向量指令进行链接，以实现并行处理技术。

例如，求向量运算 $D=A*(B+C)$ ，若向量长度 ≤ 64 ，向量的分量为浮点数，且向量 B 已取到 V_0 ，向量 C 已取到 V_1 ，分析采用下述三条指令实现功能，且采用链接技术。

$MEM \rightarrow V_3$;访存取 A 向量
$V_0 + V_1 \rightarrow V_2$;向量 B 加向量 C
$V_2 * V_3 \rightarrow V_4$; $(B+C)*A=D$

第一条向量指令与第二条向量指令既无 V_i 冲突，又无功能部件冲突，可以并行执行；第三条与第一条无功能部件冲突，与第二条也无功能部件冲突，但存在 V_i 冲突，属于源-目的向量相关，因此只要第一条指令中结果向量 V_3 的第一个分量与第二条指令中结果向量的第一个分量均产生，就可通过链接机构将这一对分量直接送往浮点乘功能部件，链接执行第三条指令。其链接过程如图 4-37 所示。本例所需拍数（亦称链接流水线的流水时间）为

$$1 \left(\begin{array}{l} \text{启动访存} \\ \text{送浮加部件} \end{array} \right) + 6 \left(\begin{array}{l} \text{访存} \\ \text{浮加} \end{array} \right) + 1 \left(\begin{array}{l} \text{存 } V_3 \\ \text{存 } V_2 \end{array} \right) + 1 \left(\begin{array}{l} \text{送浮乘部件} \\ \text{送浮乘部件} \end{array} \right) + 7 (\text{浮乘}) + 1 (\text{存 } V_4) = 17 \text{ 拍}$$

由于流水线启动后满载时，每拍可取得一个结果分量存入 V_4 。因此得到全部结果分量所需拍数为 $17 + (N-1)$ 。

3. 条件语句和稀疏矩阵加速处理方法

当程序中含有条件语句或要进行稀疏矩阵运算时，通常无法发挥向量处理优势。

例如，有下列 FORTRAN 程序：

```
DO 10 I=1,64
  IF (A(I), NE.0) THEN
    A(I) = A(I) - B(I)
  ENDIF
10 CONTINUE
```

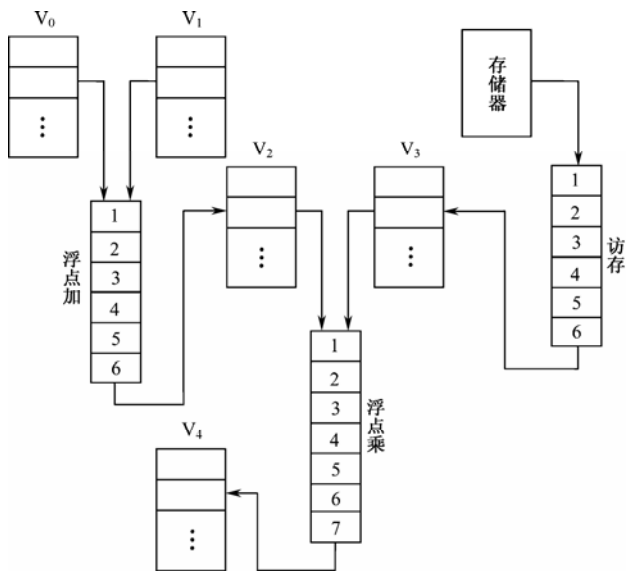


图 4-37 并行和链接操作过程图

由于要执行条件语句，因此循环体无法向量化。为加快条件语句的执行，使减法只有当 $A(I) \neq 0$ 时才执行，通常采用向量屏蔽技术实现循环的向量化。向量屏蔽技术是指用一个屏蔽向量来控制哪些向量元素参加运算，哪些向量元素不参加运算，屏蔽向量是通过测试得到的。设上例向量 A 和 B 的起始地址存放在寄存器 R_a 和 R_b 中，采用向量屏蔽技术实现上述循环程序，使用 CRAY-I 汇编语言编程：

```
LV  V1,Ra      ;向量 A 装入 V1
LV  V2,Rb      ;向量 B 装入 V2
LD  F0,#0      ;浮点数 0 装入 F0
SENSV F0,V1    ;若 V (i)  $\neq F$ ,则将  $VM_i$  置为 1
SUBV V1,V1,V2   ;在屏蔽向量控制下进行减法
CVM                    ;将屏蔽向量寄存器置为全 1
SV  Ra,V1      ;将结果存入 Ra
```

其中，SENSV 为屏蔽向量生成指令，CVM 是将屏蔽向量寄存器置为全 1 的指令。

许多元素的值为零的矩阵称为稀疏矩阵，计算机系统结构采用稀疏向量方法来解决稀疏矩阵问题。典型的稀疏矩阵运算如下：

```
DO 10 I=1,N
10 A(K(I))=A(K(I))+B(M(I))
```

该程序段是对稀疏向量 A 和 B 求和，用指标向量 K 和 M 指明 A 和 B 中的非零元素，要求 A 和 B 必须有相同的非零元素长度 N 。除了指标向量外，也可用位向量来指明非零元素。

稀疏矩阵加速处理是使用指标向量的散射-聚合 (Scatter-Gather) 操作。聚合操作是根据指标向量选取元素，其地址由基址加上指标向量中给定的相应地址偏移量而形成。聚合操作后存于目的向量寄存器中，称为稠密向量，如图 4-38 (a) 所示。散射操作是将稠密向量恢复成稀疏向量，借助同一指标向量来完成，如图 4-38 (b) 所示。

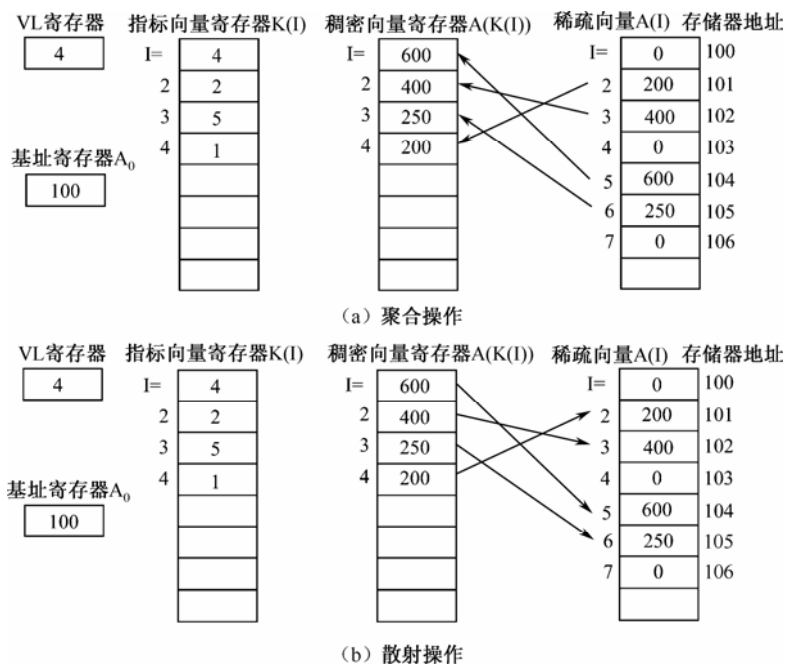


图 4-38 向量的聚合与散射操作图

4. 向量归约操作的加速方法

归约（Reduction）操作一般难以向量化。因为对一维数组的向量归约求值的结果是一个标量值。归约操作是递推（Recurrence）操作中的一个特例。向量归约操作的加速方法可采用如下两种：

- （1）将归约操作分解为可向量化部分和递推求和部分（不可向量化部分）。
- （2）在递推求和部分采用递归折叠技术，它是加快向量归约操作的有效方法。

例如，求有 64 个分量的向量 A 和 B 的点积，其 FORTRAN 程序如下：

```
DOT = 0, 0
DO 10 I = 1, 64
10 DOT = DOT + A(I) * B(I)
```

由于此循环存在迭代层间的“先写后读”的数据相关，因此无法直接向量化。现将其分解成可向量化部分和递推求和部分：

```
DO 10 I = 1, 64
10 DOT(I) = A(I) * B(I) ; 可向量化部分
DOT1 = 0, 0
DO 20 I = 1, 64
20 DOT1 = DOT1 + DOT(I) ; 递推求和部分
```

再在递推求和部分采用递归折叠技术，进行修改：

```
LEN = 32
DO 20 I = 1, 6 ; 分量个数 64=26，需递归折叠 6 次
DO 30 J = 1, LEN
```

```

30  DOT(J) = DOT(J) + DOT(J + LEN)
    LEN = LEN/2                ; 开始 32 个 PE 并行计算,
20  CONTINUE                  ; 然后 16 个 PE 并行计算……
                                ; 最后 1 个 PE 计算, 共 6 步, 即 6 次折叠。

```

对于本例向量长度为 64, 只需递推 6 步, 就可在 DOT(J)中得到所需的递推和。程序如下:

```

DO 10 I = 1, 64
10  DOT(I) = A(I) * B(I)
    LEN = 32
    DO 20 I = 1, 6
    DO 30 J = 1, LEN
30  DOT(J) = DOT(J) + DOT (J + LEN)
    LEN = LEN/2
20  CONTINUE

```

4.5.4 向量处理机的技术指标

衡量向量机性能的技术指标是向量指令流水处理时间 T_{VP} , 向量长度为无穷大时的向量机最大性能 R_{∞} 、半性能向量长度 $n_{1/2}$ 、向量和标量平衡点 n_v 。

1. 向量指令流水处理时间 T_{VP}

指在向量机上执行一条向量长度为 n 的向量指令的流水处理时间。

$$T_{VP} = T_S + T_{Vf} + (n-1) T_C$$

式中, T_S 为向量流水线的建立时间, 包括向量起始地址的设置、计数器加 1、条件转移指令执行等。 T_{Vf} 为第一对向量元素通过流水线得到结果元素的时间。 T_C 为流水线“瓶颈”段的时间。如果不存在“瓶颈”, 每段执行时间都是一个时钟周期 Δt , 则

$$T_{VP} = [s + e + (n-1)] \cdot \Delta t = (s + e - 1) \cdot \Delta t + n \cdot \Delta t$$

式中, s 为向量流水线建立时间所需时钟周期数, e 为向量流水线流过时间所需时钟周期数, n 为向量长度 (即向量元素个数), Δt 为时钟周期。令 $T_0 = (s + e - 1) \cdot \Delta t$, 则有

$$T_{VP} = T_0 + n \cdot \Delta t$$

一组向量操作的执行时间主要取决于下列三个因素: 向量的长度、向量操作之间是否存在流水功能部件冲突和数据相关性。我们把 n 条能在一个时钟周期内同时启动的向量指令称为一个编队。同一个编队内的向量指令一定不存在功能部件冲突和数据相关性。

例如, 设流水功能部件每种只有一个, 下列程序可分为多少个编队?

```

LV  V1, RX                ; 取向量 X → V1
MULTSV V2, F0, V1         ; 向量 X * 标量 F0 → V2
LV  V3, RY                ; 取向量 Y → V3
ADDV  V4, V2, V3          ; V2 + V3 → V4
SV  RY, V4                ; V4 → RY

```

第一条指令 LV 与第二条指令 MULTSV 有 V_1 相关, 所以单独列为一个编队。MULTSV 与第三条指令 LV 无关, 可列为一个编队。同理, ADDV, SV 只能各单独列为编队, 所以上述一组向量操作可划分下列 4 个编队:

- (1) LV
(2) MULTSV LV
(3) ADDV
(4) SV

一个编队执行时间为 T_{chime} ，它与向量长度无关。因此，一组由 m 个编队组成的向量操作其执行时间为 mT_{chime} 。上例中程序分为 4 个编队，所以执行时间为 $4T_{\text{chime}}$ 。如果向量长度为 n ，则整个程序执行的时钟周期 Δt 数为 mn 。全面考虑向量操作的执行时间还要加上向量启动时间 T_{start} 。 T_{start} 是流水线启动至满负荷时间，等于流水线功能段数，即流水线的深度。如果向量长度大于向量寄存器长度时，则要将向量分段执行，分段的开销由标量代码开销 T_{loop} 和每个编队的向量启动开销 T_{start} 组成。所以向量长度为 n 的一组向量操作的全部执行时间为

$$T_n = \left[\frac{n}{\text{MVL}} \right] (T_{\text{loop}} + T_{\text{start}}) + n T_{\text{chime}}$$

式中, MVL 是向量寄存器长度; T_{loop} 、 T_{start} 和 T_{chime} 的值与编译系统和处理器有关。寄存器分配和指令调度会影响编队的组合及启动开销。

例如,在某向量机上实现 $A=B*S$, 其中 A 和 B 是长度为 200 的向量, S 是标量, 向量寄存器长度为 64 位, LV 的 $T_{\text{strat}}=12$ (个时钟周期), $MULTVS$ 的 $T_{\text{strat}}=7$, SV 的 $T_{\text{strat}}=12$ 。CRAY-I 向量机标量代码开销 $T_{\text{loop}}=15$, 求总执行时间。

解：因为向量长度超过了向量寄存器长度，所以要采用分段方法。程序中循环体由下面三条向量指令组成：

LV	V_1, R_b	；取向量 B
MULTVS	V_2, V_1, F_S	；向量 B 和标量 F_S 相乘
SV	R_a, V_2	；存结果向量 A

向量 B 存于 R_b , 标量 S 存于 F_S , 结果向量 A 存于 R_a , 其他标量指令不一一列出。因为向量长度 200, 向量寄存器长度为 64 所以分段数 = $\lceil 200/64 \rceil = 4$ 。又因为上述三条向量指令存在相关性, 所以分属于三个编队, 因此, $T_{\text{chime}}=3$, 根据 T_n 公式, 则

$$T_{200}=4\times(15+T_{\text{start}})+200\times3=660+(4\times T_{\text{start}})$$

$$T_{\text{start}}=12+7+12=31$$

所以 $T_{200}=660+(4\times 31)=784$ (个时钟周期)

若每个时钟周期时间为 Δt , 则 $T_{200}=784\Delta t$ 。流水线出一个结果元素的平均执行时间(包括启动开销)为 $784/200=3.9$ (个时钟周期), 其时间为 $3.9\Delta t$ 。

2. 向量流水线的最大性能 R_{∞}

R_{∞} 表示当向量长度为无穷大时的向量流水线的最大性能，常用于评价峰值性能，单位是MFLOPS。 R_{∞} 可表示为

$$R_{\infty} = \lim_{n \rightarrow \infty} \frac{\text{浮点运算次数} \times \text{时钟频率}}{\text{循环所花费的时钟周期数}}$$

因为分子值与 n 无关, 所以

$$R_{\infty} = \frac{\text{浮点运算次数} \times \text{时钟频率}}{\lim_{n \rightarrow \infty} \text{循环所花费的时钟周期数}} = \frac{\text{浮点运算次数} \times \text{时钟频率}}{\lim_{n \rightarrow \infty} \left[\frac{T_n}{n} \right]}$$

例如, 某向量机上执行 DAXPY (double-precision $a \times X$ plus Y) 代码, 即完成 $Y = a \times X + Y$, X 和 Y 是向量, 最初存放在内存, a 是标量, 向量指令的程序如下:

LV	V_1, R_X	; 取向量 $X \rightarrow V_1$
MULTSV	V_2, F_0, V_1	; 标量 $F_0 \times$ 向量 $V_1 \rightarrow V_2$
LV	V_3, R_Y	; 取向量 $Y \rightarrow V_3$
ADDV	V_4, V_2, V_3	; 向量 $V_2 +$ 向量 $V_3 \rightarrow V_4$
SV	R_Y, V_4	; 结果向量 $V_4 \rightarrow R_Y$

因为 X, Y 在内存, 虽然第一条指令 LV 与第二条指令 MULTSV 存在 V_1 相关性, 但用链接技术 (见前述) 可实现并行, 因此构成编队 1。同理, 第三条指令 LV 与第四条指令 ADDV 使用链接技术构成编队 2。SV 为编队 3。所以 $T_{\text{chime}}=3, T_{\text{loop}}=15$, LV 的 $T_{\text{start}}=12$, SV 的 $T_{\text{start}}=12$, MULTSV 的 $T_{\text{start}}=7$, ADDV 的 $T_{\text{start}}=6$, 总的 $T_{\text{start}}=12+7+12+6+12=49$ 。向量寄存器长度 $MVL=64$, 代入 T_n 公式

$$\begin{aligned}
 T_n &= \left\lceil \frac{n}{MVL} \right\rceil [(T_{\text{loop}} + T_{\text{start}}) + n T_{\text{chime}}] \\
 &= \left\lceil \frac{n}{64} \right\rceil \times (15 + 49) + n \times 3 \\
 &= (n + 64) + 3n \\
 &= 4n + 64
 \end{aligned}$$

如设时钟频率为 200MHz, 因为程序循环体内只有 2 次浮点运算 (即 MULTSV 和 ADDV), 所以

$$R_\infty = \frac{2 \times 200\text{MHz}}{\lim_{n \rightarrow \infty} \left\lceil \frac{4n + 64}{n} \right\rceil} = \frac{2 \times 200\text{MHz}}{4} = 100\text{MFLOPS}$$

若向量流水线为线性流水线 (即无 “瓶颈” 段), 则每对向量元素的平均执行时间

$$\begin{aligned}
 t_{\text{VP}} &= \frac{T_{\text{VP}}}{n} = \frac{(s + e - 1)\Delta t + n\Delta t}{n} \\
 \lim_{n \rightarrow \infty} t_{\text{VP}} &= \lim_{n \rightarrow \infty} \frac{(s + e - 1)\Delta t + n\Delta t}{n} = \Delta t \\
 R_\infty &= \lim_{n \rightarrow \infty} \frac{1}{t_{\text{VP}}} = \frac{1}{\Delta t}
 \end{aligned}$$

线性流水线的最大性能为 $1/\Delta t$ 。 Δt 是每个功能段的时间。

3. 半性能向量长度 $n_{1/2}$

$n_{1/2}$ 是为达到一半 R_∞ 值所需的向量长度, 它是评价向量流水线建立时间对性能影响的参数, 反映建立流水线而导致的性能损失。若向量长度 $n = n_{1/2}$, 表明整个向量流水处理时间中只有一半时间在做有效操作, 而另一半是浪费掉的, 一般希望 $n_{1/2}$ 较小。实测表明, CRAY-1 向量处理机的 $n_{1/2}=10 \sim 20$, CYBER205 的 $n_{1/2}=100$, 反映出 CRAY-1 流水线建立时间比 CYBER205 小得多。

例如, 在上例中 200MHz 的向量机上执行 DAXPY 程序, $R_\infty = 100\text{MFLOPS}$, 则其 $n_{1/2}$ 为多少?

由 MFLOPS 定义可知: $n_{1/2}$ 时 MFLOPS 定义为

$$\frac{\text{执行 } n_{1/2} \text{ 循环时的浮点运算次数}}{\text{执行 } n_{1/2} \text{ 循环的时钟周期数}} \times \frac{\text{时钟周期}}{\text{秒}} \times 10^{-6}$$

因为 $R_{\infty} = 100\text{MFLOPS}$ ，所以， $\frac{1}{2}R_{\infty} = 50\text{MFLOPS}$

$$50 = \frac{2 \times n_{1/2}}{T_{n_{1/2}}} \times 200$$

$$T_{n_{1/2}} = 8 \times n_{1/2}$$

设 $n_{1/2} \gg 64$ ，则向量处理时间 $T_{n < 64} = 1 \times 64 + 3n$

可得 $1 \times 64 + 3 \times n_{1/2} = 8 \times n_{1/2}$

所以 $n_{1/2} = 12.8 \gg 13$

4. 向量和标量的平衡点 n_v

n_v 表示向量流水方式的工作速度优于标量串行工作时所需向量长度临界值。该参数既衡量建立时间，也衡量标量、向量速度比对性能的影响。上例 $n_v < 64$ ，如果采用标量方式工作，一个循环执行时间为 $10+12+12+7+6+12=59$ 个时钟周期，其中 10 是标量建立循环开销，以后分别为取向量 X、取向量 Y、乘法、加法、写结果的开销。如果按向量方式工作，则执行时间

$$T_{n < 64} = 64 + 3n$$

因此 $64+3n_v=59n_v$

$$n_v = \left\lceil \frac{64}{56} \right\rceil = 2$$

4.5.5 多向量多处理机概述

1. 向量处理机的历史和现状

20 世纪八九十年代由美国和日本制造的大规模超级向量流水处理机如表 4-1 所示。

表 4-1 美国和日本制造的大规模超级计算机一览表

系 统 型 号	最大配置、时钟周期、操作系统/编译系统	特色和要点
CRAY 1S	有 10 条流水线的单处理机 12.5ns, COS/CF 77 2.1	第一台基于 ECL 的超级计算机，1979 年问世
CRAY 2S 4-256	256MB 存储器的 4 台处理机 4.1ns, COS 或 UNIX/ CF 77 3.0	16KB 的本地存储器，移植了 UNIX V, 1985 年问世
CRAY X-MP 416	16MB 存储器的 4 台处理机，128MB SSD, 8.5ns, COS CF77 5.0	使用共享寄存器组用于 IPC, 1983 年问世
CRAY Y-MP 832	128MB 存储器的 8 台处理机，6ns, CF77 5.0	X-MP 的改进型，1988 年问世
CRAY Y-MP C-90	每台处理机 2 条向量流水线，16 台处理机，4.2ns, UNICOS/CF77 5.0	最大的 CRAY 机器，1990 年问世
CDC Cyber 205	有 4 条流水线的单处理机，20ns, 虚拟 OS/FTN 200	存储器到存储器系统结构，1982 年问世
ETA 10E	单处理机，10.5ns, ETAV/FTN 200	Cyber 205 的后继型号，1985 年问世
NEC SX-X/44	每台处理机 4 组流水线，4 台处理机，2.9ns, F77SX	1991 年问世
Fujitsu VP 2600/10	5 条流水线的单处理机和双标量处理机，3.2ns, MSP.EX/F77 EX/VP	使用可重构向量寄存器和屏蔽，1991 年问世
Hitachi 820/80	512MB 存储器，18 个流水线功能部件的单处理机，4ns, FORT 77/HAP V23-OC	64 个 I/O 通道，最大传输速率为 288MB/s, 1988 年问世

(1) CRAY 系列。CRAY-1 是 1975 年问世的，它的改进型 CRAY-1S 是 1979 年生产的，是第一台采用 ECL 半导体工艺制造的芯片，时钟周期为 12.5ns（80MHz）向量机。高度流水线和向量处理是它的特点。CRAY-1S 有 10 个流水线功能部件，可以同时运行，其计算能力相当于 10 台 IBM 3033 或 10 台 CDC Cyber 7600。CRAY-1 使用 CRAY 公司的操作系统(COS)和 FORTRAN77 编译器（CF 77 2.1 版），只允许单用户进行批处理。1983 年 CRAY X-MP 系列采用多处理机结构，用 1~4 个 CRAY-1 CPU 和共享存储器开发，采用共享寄存器组，加快了处理机之间的通信。共享存储器有 128MB，还有 1GB 的固态存储设备(Solid-state Storage Device, SSD)作为扩展的共享存储器，时钟周期 8.5ns（117.6MHz）。当 4 台处理机同时使用 8 条向量流水线进行加和乘时，X-MP-416 峰值速度可达 840MFLOPS。CRAY 公司 1988 年生产了 CRAY Y-MP，单个系统最多有 8 台处理机，时钟周期 6ns（166.7MHz），共享存储器为 256MB。1990 年生产了 CRAY Y-MP C-90，它是一个集成系统，有 16 台处理机，时钟周期 4.2ns（238MHz）。1985 年推出的 CRAY-2S，该系统有 4 台处理机，2GB 共享存储器，时钟周期为 4.1ns（244MHz）。CRAY-2S 的主要贡献是，使超级计算机的批处理 COS 改变到多用户 UNIX 系统 V。目前大多数 CRAY 机上的 UNICOS 操作系统都是由 UNIX V 和 Berkeley 4.3 BSD 演变而来的。

(2) Cyber/ETA 系列。CDC 公司于 1973 年推出第一台向量机 STAR-100 后，于 1982 年生产了 Cyber 205。Cyber 205 单处理机配置，内有 4 条向量流水线，时钟周期 20ns（50MHz）。Cyber 205 及其后继的 ETA10E 采用“存储器-存储器”的系统结构，向量指令比较长，其中包含了存储器地址。ETA10E 由 8 个 CPU 和共享存储器以及 18 台 I/O 处理机组成。ETA10E 峰值速度是 10GFLOPS。

(3) 日制向量计算机系列。NEC 公司生产的 SX-X 系列于 1991 年推出，宣称峰值速度达到 22GFLOPS。时钟周期为 2.9ns（345MHz）。Fujitsu 公司生产的 VP 2000 系列的峰值速度达 5GFLOPS，时钟周期为 3.2ns（312.5MHz）。Hitachi 公司的 820 系列机的峰值速度为 3GFLOPS。这些机器的主要特点是，采用共享通信寄存器和可重构的向量寄存器。日制向量机在高速硬件和交互式向量化编译器方面是较先进的。

表 4-2 概括了三种有代表性的多向量处理机系统结构的特性。

表 4-2 三种多向量处理机系统结构的特性

机 器 特 性	CRAY Y-MP C90/16256	NEC SX-X 系列	Fujitsu VP 2000 系列
处理机台数	16 个 CPU	4 台运算处理机	VP2600/10: 1 台处理机 VP2400/40: 2 台处理机
机器周期时间	4.2ns	2.9ns	3.2ns
最大存储器容量	256 兆字（2GB）	2GB，1024 路交叉访问	1GB 或 3GB 的 SRAM
可选的 SSD 存储器	512 兆、1024 兆或 2048 兆字（16GB）	16GB，传输速率为 2.75GB/s	32GB 的扩充存储器
处理机系统结构：向量流水线、功能部件和标量部件	每个 CPU 有两条流水线和两个功能部件，每个时钟周期发送 64 个向量结果	每台处理机有 4 组向量流水线，每组有两条加法/移位流水线和两条乘法/逻辑流水线，一条标量流水线	每个向量部件有两条装入/存储流水线和 5 个流水线功能部件。1~2 个向量部件。每个向量部件可以和 2 个标量部件相连
机器特性	CRAY Y-MP C90/16256	NEC SX-X 系列	Fujitsu VP 2000 系列

续表

机 器 特 性	CRAY Y-MP C90/16256	NEC SX-X 系列	Fujitsu VP 2000 系列
操作系统	由 UNIX/V 和 BSD 演变而来的 UNICOS	基于 UNIX 系统 V 和 4.3BSD 的 Super-UX	UPX/M 和用于向量处理的 MSP/EX
前端机	IBM, CDC, DEC, Univac Apollo, Honeywell	内部控制处理器和 4 台 I/O 处理机	与 IBM 兼容的主机
向量化语言/编译器	FORTRAN 77, C, CF 77 5.0, CRAY C 3.0 版本	FORTRAN 77/SX, 向量化器, 分析器/SX	FORTRAN 77 EX/VP, 有交互式向量化器的 C/VP 编译器
峰值性能和 I/O 频宽	16GFLOPS 13.6GB/s	22GFLOPS, 每台 I/O 处理机 1GB/s	5GFLOPS, 256 个通道, 2GB/s

2. 典型向量处理机系统结构简介

(1) CRAY Y-MP 和 C-90

CRAY Y-MP 和 C-90 是 CRAY 公司具有代表性的向量机,CRAY Y-MP 系统结构框图如图 4-39 所示。

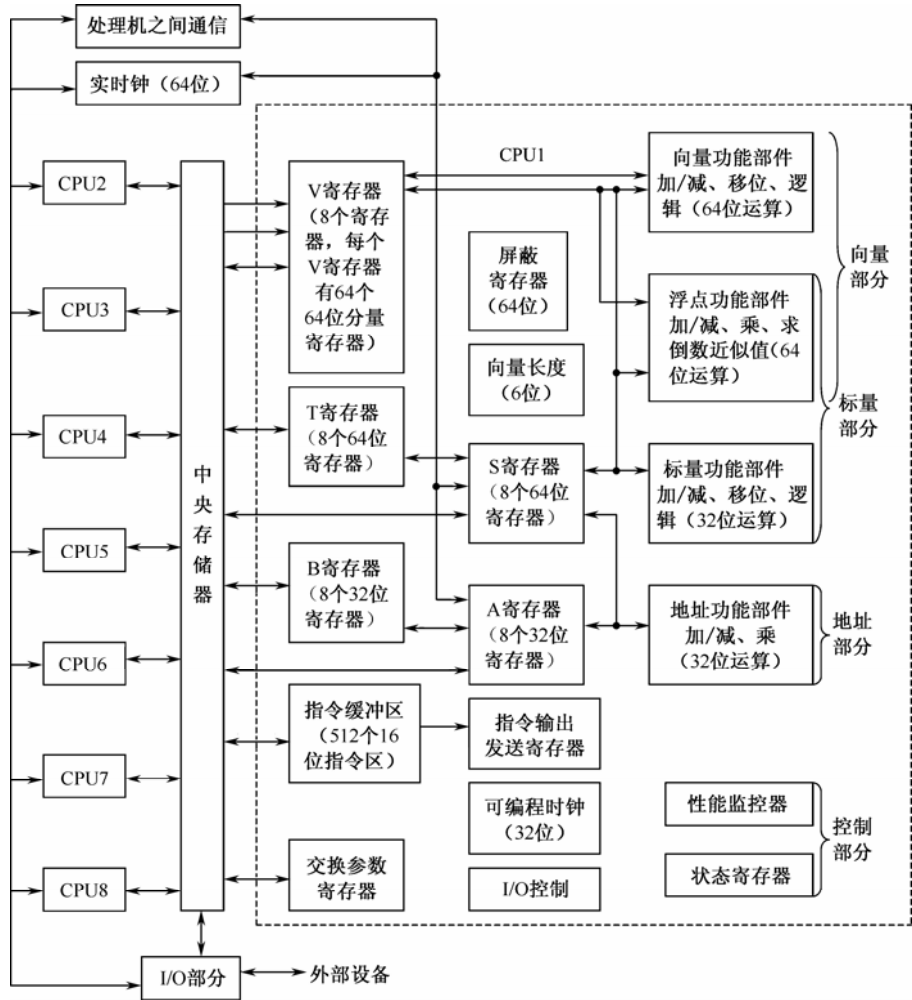


图 4-39 CRAY Y-MP 系统结构框图

系统可以配置 1, 2, 4 和 8 台处理机, Y-MP 的 8 个 CPU 共享中央存储器、I/O 子系统、处理机通信子系统和实时钟。中央存储器由 256 个交叉访问的存储体构成。每个 CPU 对 4 个 MEM 端口的交叉访问可以实现对 MEM 的重叠存取, CPU 的时钟周期为 6ns (166.7MHz)。中央 MEM 的容量可以是 128MB, 256MB, 512MB, 1024MB, 最大可达 1GB。固态 MEM 容量可以从 256MB~4096MB, 最大可达 4GB。4 个 MEM 访问端口允许每个 CPU 同时执行两个标量和向量取操作, 一个存储操作和一个独立的 I/O 操作。这些并行的 MEM 访问采用流水线方式, 使向量读和写同时进行。系统内部有化解冲突的硬件, 使 MEM 冲突引起的延迟减到最小。在中央 MEM 及其 I/O 数据通道中都采用单错校正/双错检测 (SECDED) 逻辑, 用于保护数据。CPU 内的计算系统由 14 个功能部件组成, 分为标量、向量、地址和控制 4 个子系统, 如图 4-39 所示。向量和标量指令可以并行执行。向量指令使用 14 个中的 8 个功能部件。所有算术运算都是“寄存器-寄存器”类型。系统使用大量的寄存器, 有地址 Reg、标量 Reg、向量 Reg、中间 Reg 和临终 Reg (Reg 即寄存器)。通过对 Reg 及多条 MEM 和算术/逻辑流水线的使用, 可灵活实现不同功能流水线的链接。浮点和整数算术运算都是 64 位。指令 Cache 可存放 512 条 16 位的指令。CPU 之间通信系统包括用于快速同步目的的共享 Reg 群, 每个群由共享地址 Reg、共享标量 Reg 和信号灯 Reg 组成。CPU 之间向量数据的通信是通过共享 MEM 实现的。实时钟由 64 位计数器组成, 每个时钟周期计数器加 1。由于时钟与程序执行同步, 所以它可以准确地计算时间。I/O 子系统支持三类通道, 传输速率分别为 6MB/s, 100MB/s 和 1GB/s。IOS 和 SSD (固态存储器设备) 是高速数据传输设备, 通过 8 个高速缓存支持主机的处理工作。

C-90 对 Y-MP 系列在技术和规模上做了进一步改进。系统由 16 个类似于 Y-MP 的 CPU 组成。16 台处理机共享的主存容量达 2GB。SSD 存储器的容量最长达 16GB, 可选作第二级主存。两条向量流水线和两个功能部件可以并行操作, 每个时钟周期产生 4 个向量计算结果, 即每台处理机有 4 路并行。因此 16 台处理机每个时钟周期最多可以产生 64 个向量计算结果。C-90 运行 UNICOS 操作系统, 它是 UNIX V 和 Berkeley BSD 4.3 经过扩充而成的。该系统提供向量化的 FORTRAN 77 和 C 编译器。64 路并行性和 4.2ns (238.1MHz) 时钟周期配合, 可使系统峰值达到 16GFLOPS, 系统最大 I/O 频宽为 13.6MB/s, 许多主机可驱动 C-90。

为了提升计算能力, 求解大型问题, 可以把多台 C-90 连成机群结构, 如图 4-40 所示, 4 个 C-90 机通过速率为 1000MB/s 的通道与 SSD 连接, 每个 C-90 只能访问自己的主存, 然而它们共享 SSD。每个 C-90 可以通过共享信号灯部件与其他 C-90 通信, 只有同步和控制信息才通过信号灯部件传输。因此, C-90 机群是松散耦合系统, 提供的最大并行性为 256 路 (每个 C-90 为 64 路并行, 共 4 台 C-90)。如果任务分配很好并且机群间负载很均衡, 其峰值可达到 64GFLOPS。

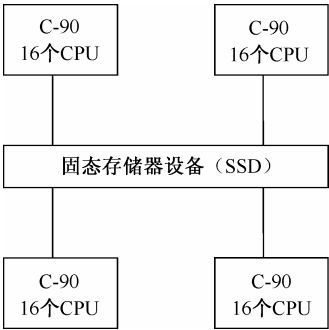


图 4-40 与公共 SSD 相连的 4 台 CRAY Y-MP C-90 构成的 256 路并行系统

(2) Fujitsu VP 2000 和 VPP 500

VP 2000 系列是日本 Fujitsu 公司制造的多向量多处理机，系统可配置 1 台或 2 台处理机。VP 2000 系列超级计算机系统结构如图 4-41 所示，它可以扩展成双处理机系统 VP 2400/40。系统时钟为 3.2ns(312.5MHz)，主存容量为 1GB 或 2GB，系统存储器可扩充到 32GB。VP 2000 中每个向量部件由两条装入/存储流水线、三条功能流水线和两条屏蔽流水线组成，两个标量部件与一个向量部件相连。在双处理机配置时，可以有 4 个标量部件和两个向量部件。VP 2000 系列（有 10 个型号）最大向量峰值范围为 0.5~5GFLOPS。VP 2000 中有总数为 8K 个分量寄存器，每个寄存器 64 位。寄存器堆可重构成 8, 16, 32, 64, 128, 256 个向量寄存器，分别有 1024, 512, 256, 128, 64, 32 个分量寄存器。

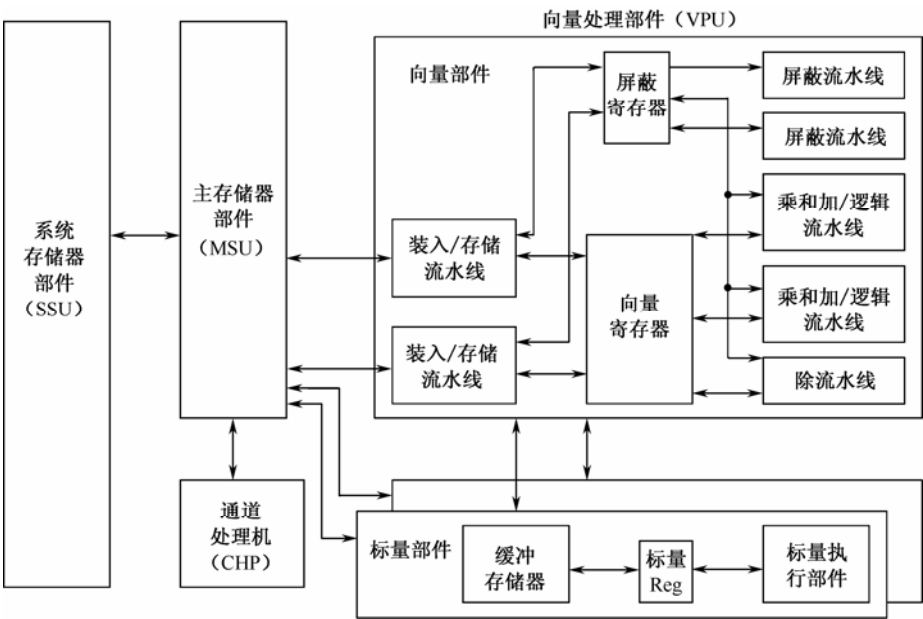


图 4-41 Fujitsu 公司的 VP 2000 超级计算机系统结构框图

VPP 500 是 Fujitsu 公司推出的向量并行处理机 (Vector Parallel Processor)，它可以配置 1 台或 2 台控制处理机，7~222 个处理部件 (PE)，有一个高性能的分布式存储系统，对 MIMD 提供灵活有效的支持。系统使用全局共享虚拟主存的方法，将所有 PE 的局部 MEM 在系统范围内进行统一编址，使 PE 间可相互进行地址访问。图 4-42 是该机的系统结构框图。VP 2000 或 VPX 200 为前台机，VPP 500 为后台机。VPP 500 系列计算机系统融合了当前 MPP（大规模并行处理机）领域许多新技术，如交叉开关、路障 (Barrier) 同步机制和系统分区机制等。VPP 500 系统的 PE 直接通信是经过 224×224 无冲突高速交叉开关互连网络实现的，采用 TCP/IP 协议进行 PE 间的进程通信，有效地提高了数据传输速率，使用户程序能够直接进行数据传送。路障同步机制可以用硬件对运行在不同 PE 上的进程进行同步控制，大幅减少操作系统开销。只有当被同步的所有进程都到达同步点后，进程才继续往下执行。VXP/VPP 操作系统将所有 PE 分成若干不同的 PE 组，在作业运行时，系统根据各作业对资源的要求将作业分配到不同的 PE 组执行。系统的每个分区独立地执行自己作业队列中的作业，这种分区机制确保了分布式存储结构下的系统灵活性。PE 由标量部件 (Scalar Unit, SU)、向量部件

(Vector Unit, VU)、数据传输部件 (Data Transfer Unit, DTU) 和局部主存组成, 是一个独立的计算节点, PE 结构如图 4-42 右面所示。标量部件采用长指令字 (Long Instruction Word, LIW) RISC 技术, 在单时钟周期内并发执行三个操作, 峰值性能为标量运算 300MIPS, 浮点运算 200MFLOPS。向量部件有一个 128KB 的大容量向量寄存器、向量屏蔽寄存器和 7 条流水线, 峰值性能达到 2.2GFLOPS。PE 内的浮点运算、内存访问和向量指令执行是异步的, 通过专门的数据传输部件, PE 可以同时传送、接收和处理数据。DTU 单向传输速率为 400MB/s, 双向传输速率为 800MB/s, DTU 还将逻辑地址转换成物理地址, 以利于访问虚拟全局 MEM。DTU 还有专用硬件实现快速路障同步机制。每个 PE 的局部 MEM 最多可扩展到 1GB, 峰值性能达到 1.6~2.2GFLOPS。VPP 500 系列 PE 数有 4~224 个 (含两个控制处理机 CP), 最高性能可达 355.2GFLOPS, 总存储容量可达 222GB。

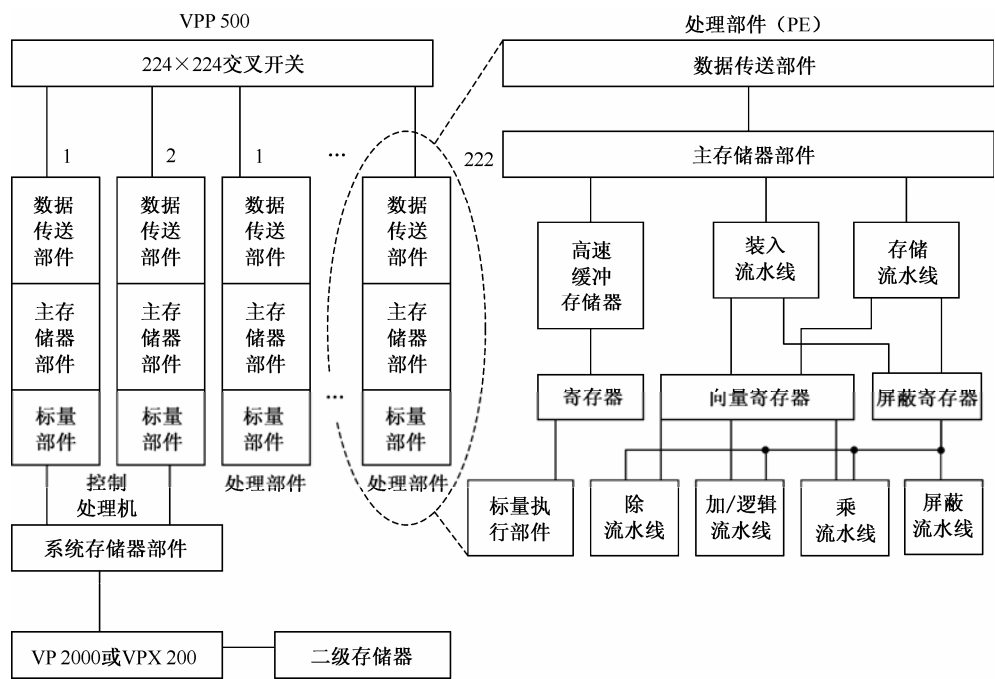


图 4-42 Fujitsu 公司的 VPP 500 超级计算机系统结构框图

(3) NEC SX-X44

1991 年日本 NEC 公司生产的 SX-X44 系列机, 使用了 VLSI 和高密度封装技术, 时钟周期 2.9ns (345MHz), 其系统结构框图如图 4-43 所示。4 台运算处理机 (AP₀~AP₃) 通过共享寄存器或通过 2GB 共享存储器进行通信。每台处理机有 4 组向量流水线, 每组有 2 条加法/移位流水线和 2 条乘法/逻辑流水线, 共有 16 条流水线。因此, 系统最多可达 64 路并行性。系统的高速标量部件采用有 128 个标量寄存器的 RISC 系统结构。主 MEM 为 1024 路交叉访问 MEM, 扩展 MEM 容量可达 16GB, 最大传输速率为 2.75GB/s。通过指令重新排序可以开发较高并行性。系统最多可配置 4 台 I/O 处理机, 每台 I/O 处理机的数据传输速率为 1GB/s。系统能支持 100MB/s 速率的通道工作, 最多可提供 256 个通道, 用于高速网络、图形和外围操作。

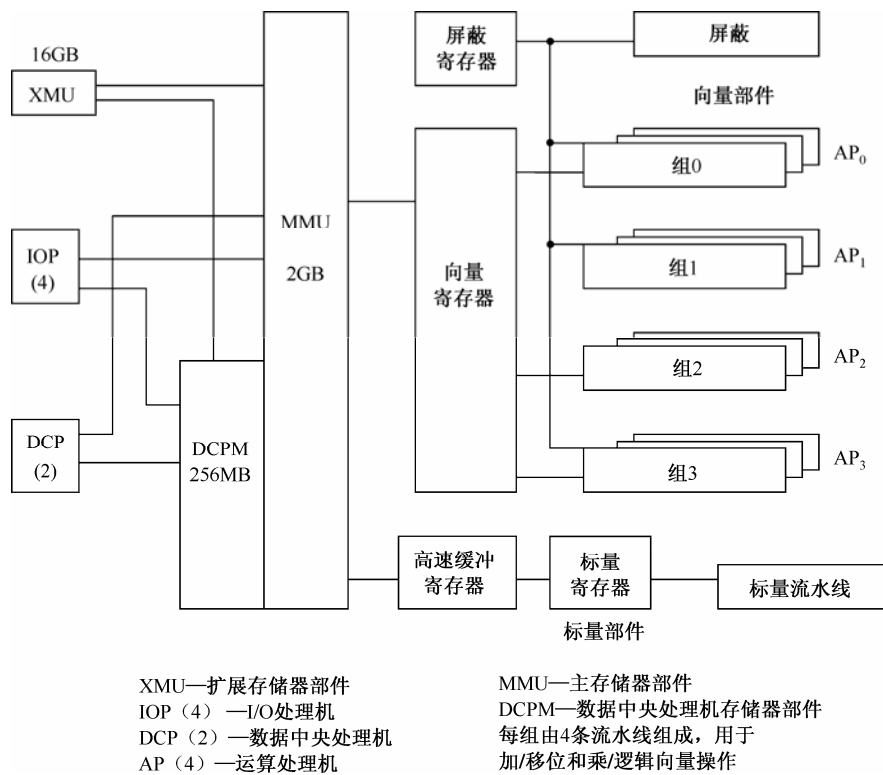


图 4-43 NEC 公司的 SX-X44 向量处理机系统结构框图

3. 向量处理机的评价

向量计算机系统结构的主要优点是：

- (1) 存储器使用流水存取方式，充分使用存储器带宽。
- (2) 流水结构的运算器有很高的性能价格比。
- (3) 通信和同步的机制很简单。

向量计算机系统结构采用下述技术措施解决技术问题：

- (1) 处理器带宽。运算器采用流水线结构，或者用多个运算器构成全并行结构。大型向量机将两者结合起来，即多条流水线构成全并行结构。
- (2) 存储器带宽。采用多个存储体构造大容量多体交叉并行存储系统，其带宽随存储体数目增加而增大，为了和运算器带宽匹配，可以采用多层次存储系统，其中 Cache 和可寻址寄存器组的效果最好，发展趋势是 Cache 容量越来越大。在存储系统中采用流水线技术，存储速率比传统的存储系统快 5~20 倍。所以与运算器内流水技术配合，是打破计算机内两个速度瓶颈的有效措施。
- (3) I/O 带宽。随着存储系统带宽增加，I/O 带宽也增大，许多高性能系统都配备 10~20 个 DMA 通道，其速度与主存速度相匹配。
- (4) 通信带宽。大多数向量机不需要处理器与处理器之间进行通信。信息分布在向量运算的各操作数之中，信息也可以通过共享存储器分布到不同的向量上，通信带宽与存储器带宽是一致的。
- (5) 同步。对于单条流水线，运算按进入流水线的顺序执行，所以同步是自动进行的。

对于多条流水线结构，可以用一个控制程序使所有流水线同步工作（如 FPS164），也可采用流水线互锁控制向量操作，使不冲突的操作并行执行，相关操作尽可能链接起来重叠地进行（如 CRAY 1）。

（6）多用途。向量机支持多种存取数据方式，并且有丰富的向量指令，对解决大多数向量问题都是有效的。但应用仅局限于数值计算。

向量计算机系统发展趋势如下：

- （1）提供更多的向量指令。
- （2）除向量处理功能外还扩展其他功能。
- （3）采用多层次存储系统。
- （4）流水线技术与并行技术相结合。

4.6 超级流水处理机

本节主要介绍超标量处理机（Superscalar Processor）、超流水线处理机（Superpipelining Processor）、超长指令字处理机（Very Long Instruction Word Processor）、超标量超流水线处理机（Superscalar Superpipelining Processor）的基本原理、典型结构和主要性能等。

以一台 k 段流水线的普通标量处理机为基准，与上述三种处理机（除超长指令字处理机）主要性能比较如表 4-3 所示。

表 4-3 4 种不同类型处理机的性能比较

机器类型	k 段流水线 基准标量处理机	m 度超标量处理机	n 度超流水线处理机	(m, n) 度超标量 超流水线处理机
机器流水线周期	1 个时钟周期	1	$1/n$	$1/n$
同时发射指令条数	1 条	m	1	m
指令发射等待时间	1 个时钟周期	1	$1/n$	$1/n$
指令级并行度 ILP	1	m	n	$m \times n$

在表 4-3 中，基准标量处理机是一台普通单流水线处理机，其机器流水线周期、同时发射指令条数、指令级并行度 ILP（Instruction Level Parallelism）均假设为 1，则并行度为 m 的超标量处理机、并行度为 n 的超流水线处理机、并行度为 (m, n) 的超标量流水线处理机的性能相对与基准标量处理机进行比较后的结果在表 4-3 中已列出。

4.6.1 超标量处理机

超标量（Superscalar）机器是为改善标量指令执行性能而设计的机器，是高性能通用处理器发展的一个方向，其本质是提高在不同流水线中执行不相关指令的能力。超标量处理机中使用了多指令流水线，每个时钟周期发射多条指令并产生多个结果。因此，对用户程序开发更多的指令级并行性是设计超标量处理机必须考虑的因素之一，而只有不相关指令才能并行执行而不相互等待。指令级并行性的变化与执行代码的类型有很大关系。单发射基准流水线时空图如图 4-44 所示。由于复杂指令往往需要多个时钟周期才能完成，另外还有条件转移等影响，所以一般流水线标量处理机每个时钟周期平均执行指令条数小于 1，即指令级并行

度 $ILP < 1$ 。

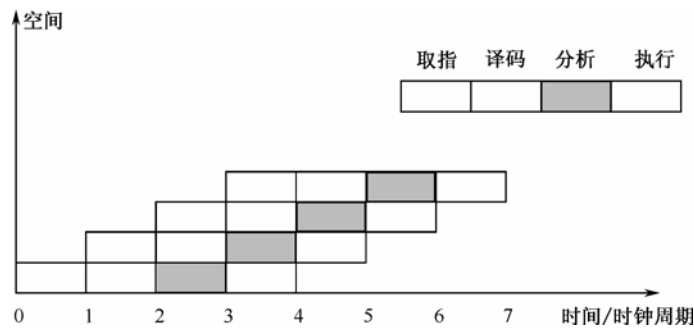


图 4-44 单发射基准流水线的流水操作

经统计发现，对于没有循环展开的指令代码，指令级并行性的平均值 $ILP \approx 2$ 。每个周期发射指令超过三条的机器并没有更高的 ILP 。所以在超标量处理机中，指令发射度实际上限制在 2~5 之间。图 4-45 为并行度为 3 的超标量流水线的时空图。超标量流水线是指每个时钟周期同时发射多条指令并产生多个结果的流水线。超标量方法的实现取决于指令并行执行的程度，主要借助硬件资源重复工作来实现空间的并行操作。超标量处理器的特点如下：

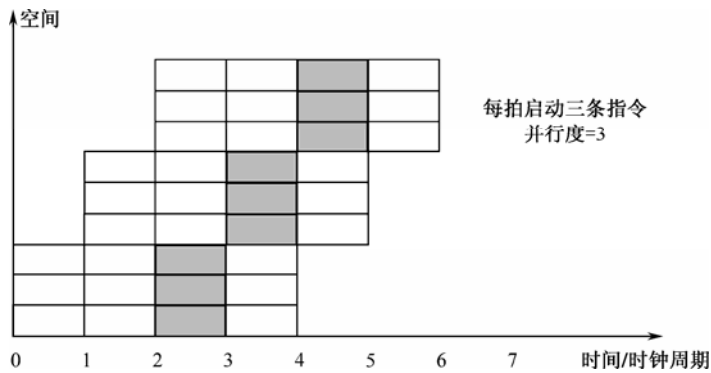
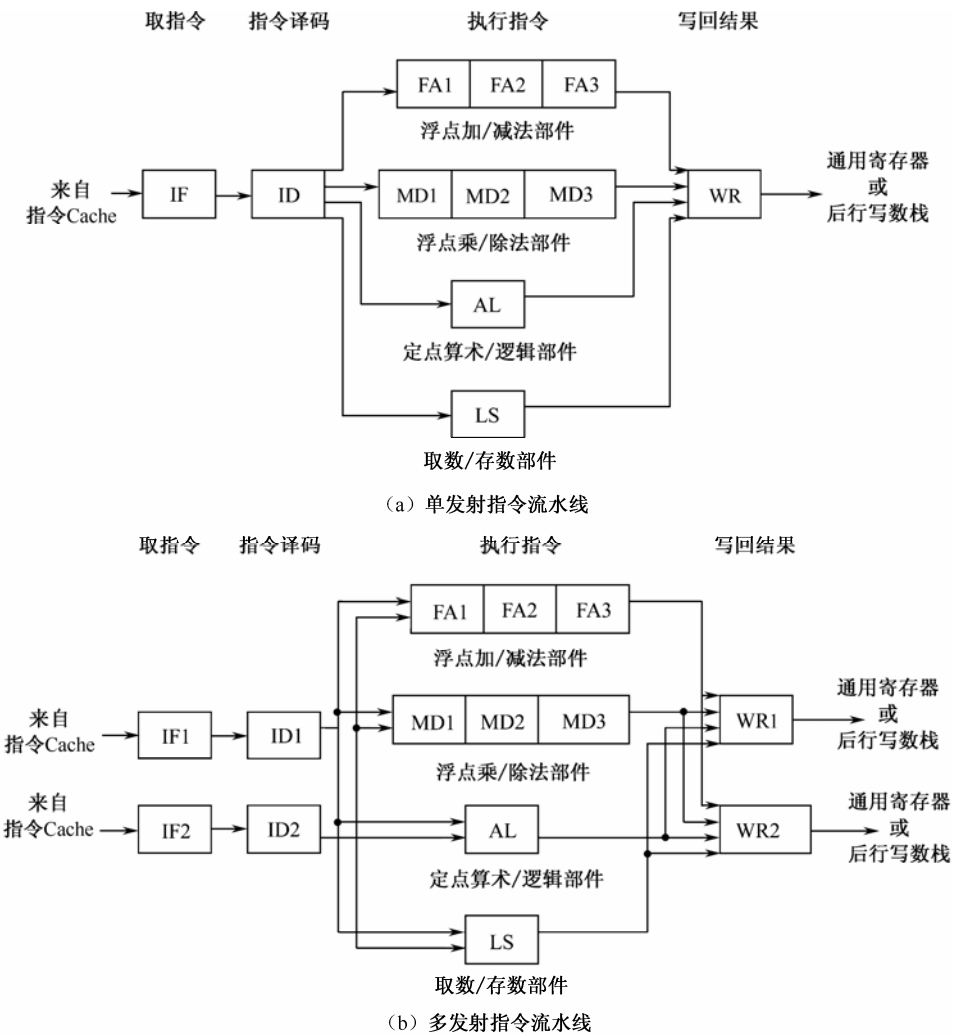


图 4-45 三发射超标量流水线

- (1) 配置多个性能不同的处理部件，采用多条流水线并行处理。
- (2) 同时对若干条指令进行译码，将可执行的指令送往不同部件，达到在一个时钟周期内启动多条指令的目的。

(3) 在程序执行期间由硬件（状态记录部件和调度部件）完成指令调度。

超标量处理机的典型结构是有多个操作部件，一个或几个较大的通用寄存器堆，一个或两个高速 Cache，一般有三个处理单元：第一个是定点处理单元，由一个或多个处理部件组成，称为中央处理单元（CPU）；第二个是浮点处理单元（FPU），由浮点加/减法部件和浮点乘/除法部件组成；第三个是图形加速部件（GPU），这是现代处理机中不可缺少的部分。CPU 和 FPU 分别使用两个通用寄存器堆，有的还采用 RISC 中的寄存器窗口技术。Cache 采用哈佛结构，指令 Cache 和数据 Cache 分开，各有几 KB 至几十 KB 容量。有的超标量处理机把二级 Cache 也做在芯片内。超标量处理机指令的流水线结构如图 4-46 所示。



FA—浮点加/减法运算；MD—浮点乘/除法运算；AL—定点算术/逻辑运算；LS—取数/存数

图 4-46 超标量处理机的指令流水线

超标量处理机的 $ILP > 1$ ，如果每个时钟周期发射 m 条指令，则 ILP 期望值就为 m 。但是由于数据相关、条件转移和资源冲突等原因，实际的 ILP 不可能达到 m ，通常为 $1 < ILP < m$ 。在超标量处理机中，不仅有多套取指部件和指令译码部件，而且要判断指令间有无功能部件冲突、有无数据相关和条件转移引起的控制相关等，同时还要有一套交叉开关把几个指令译码器的输出送到多个操作部件去执行。因此超标量处理机的控制逻辑比较复杂。当出现相关时，本次没有发射出去的指令必须保存下来，以便在下一个周期时再发射。为此，设置一个先行指令窗口，在窗口中保存由于各种相关而不能送出执行的指令，其典型结构如图 4-47 所示。该窗口的作用类似于先行控制技术中的先行指令缓冲栈，可以从指令 Cache 中读入更多的指令，通过硬件判断将没有冲突、没有相关的指令超越前面的指令先发射到操作部件中去，从而提高功能部件的利用率。如果再加上编译器支持，将没有冲突和相关或者冲突和相关较少的指令调度到同一个先行指令窗口中，就能够进一步提高超标量处理机的性能。窗口大小对超标量处理机性能影响很大：窗口太小，调度效果不好；窗口太大，则调度的硬件太复杂。

一般指令窗口大小为 2~8 条指令。

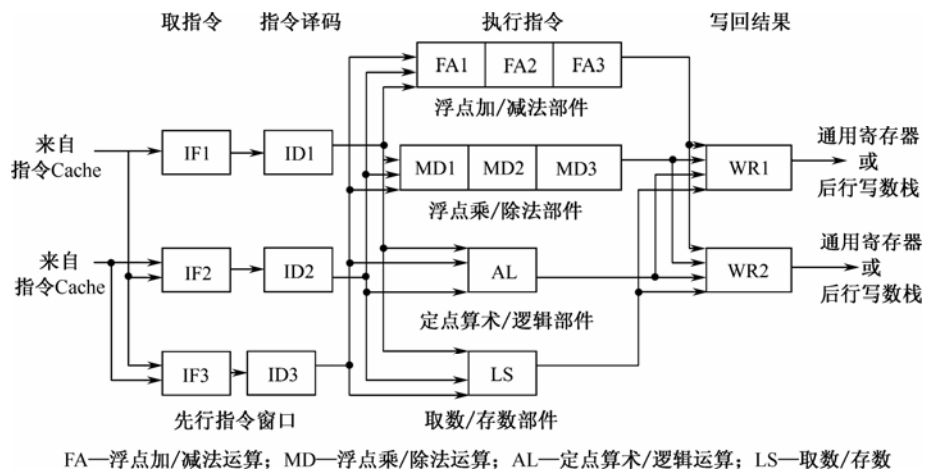


图 4-47 有先行指令窗口的多发射流水线处理机结构

为了便于比较，把单流水线标量处理机的指令级并行度 ILP 记作 (1, 1)，超标量处理机 ILP 记作 (m, 1)，超流水线处理机 ILP 记作 (1, n)，超标量超流水线处理机 ILP 记作 (m, n)。

在无冲突、无相关的前提下，N 条指令在单流水线标量处理机上的执行时间为

$$T(1, 1) = (K + N - 1) \cdot \Delta t$$

式中，K 是流水线级数，Δt 是时钟周期时间。如果把 N 条指令在一个 Δt 内发射 m 条指令的超标量处理机上执行，所需要的时间为

$$T(m, 1) = (K + \left\lceil \frac{N - m}{m} \right\rceil) \cdot \Delta t$$

式中，第一项是第一批 m 条指令同时通过 m 条流水线所需要的执行时间，第二项是余下的 N - m 条指令所需要的时间，此时每个 Δt 有 m 条指令分别通过 m 条流水线。因此超标量处理机的加速比为

$$S(m, 1) = \frac{T(1, 1)}{T(m, 1)} = \frac{m(K + N - 1)}{N + m(K - 1)}$$

当 N → ∞ 时，在无冲突、无相关的理想情况下，超标量处理机的加速比最大值为

$$S(m, 1)_{\max} = m$$

典型的超标量处理机有 IBM RS6000，Power PC601，Power PC620 等，Intel 公司的 i860，i960，Pentium 系列微处理器，SUN 公司的 Ultra SPARC 系列微处理器，Motorola 公司的 MC88110 等。MC88110 结构如图 4-48 所示。它有 10 个操作部件，其中两个整数部件可作 32 位的整数运算、地址运算等，使用一条 4 功能段流水线，单周期执行。浮点运算字长为 80 位，可完成浮点加、减、乘、除和求浮点平方根。浮点加法部件和乘法部件都采用三级流水线，每个时钟周期完成一条乘法指令和一条浮点加法指令。两个专用图形部件可以直接对图形像素进行处理，提供三维 (3D) 图形处理能力。整数部件使用通用寄存器堆，由 32 个 32 位寄存器组成。浮点部件使用扩展寄存器堆，由 32 个 80 位寄存器组成。每个寄存器堆有 8 个端口，分别与 8 条内部总线相连，可以同时读出 8 个数据提供给各操作部件使用。在读数/存数部件内有一个缓冲深度为 4 的采用先进先出 (FIFO) 方式的先行读数栈和一个缓冲深度

关的指令所需时间为工业

$$T(1, n) = (K + \frac{N-1}{n}) \cdot \Delta t$$

式中， K 是指令流水线的功能段数或时钟周期数，而不是流水线级数。如图 4-49 中功能段为 4 个，即 $K=4$ 。指令流水线的级数实际应为 Kn 。如图 4-49 中 $n=3$ ，则超流水线级数为 $4 \times 3=12$ 级。上式中第一项是第一条指令执行完成所需时间，第二项是执行其余 $(N-1)$ 条指令所需要的时间，此时每一个时钟周期有 n 条指令执行完成。超流水线处理机相对于单流水线普通标量处理机加速比为

$$S(1, n) = \frac{T(1,1)}{T(1,n)} = \frac{n(K+N-1)}{nK+N-1}$$

当执行的指令数 $N \rightarrow \infty$ 时，在没有数据相关和控制相关的理想情况下，超流水线处理机的加速比最大值为

$$S(1, n)_{\max} = n$$

超流水线处理机典型产品有 CRAY-1 型和 CDC-7600，其指令级并行度 $n=3$ 。目前只有 SGI 公司的 MIPS (Microprocessor without Interlocked Piped Stages) 系列处理机属于超流水线处理机。图 4-50 所示为 MIPS R4000 结构。R4000 内有指令 Cache 和数据 Cache，容量各为 8KB。每个 Cache 的数据宽度为 64 位，每个时钟周期可以访问两次 Cache，即可以从指令 Cache 内取出两条指令，从数据 Cache 中存取两个数据。整数部件包括 32 个 32 位通用寄存器、ALU、专用乘/除法部件。寄存器堆有两个输出端口和一个输入端口，有专用的数据通路，可以对每个寄存器读/写两次。ALU 负责算术运算、地址运算、逻辑运算和移位操作。乘/除法部件能够执行 32 位有符号和无符号乘/除法，与 ALU 并行执行指令。浮点部件由浮点通用寄存器堆和执行部件组成，浮点通用寄存器堆由 16 个 64 位寄存器组成，也可设置成 32 个 32 位浮点寄存器。浮点执行部件由浮点乘法、浮点除法、浮点加法/转换/求平方根等三个独立部件组成，可以并行工作。

R4000 的指令流水线有 8 级，流水线操作如图 4-51 所示。R4000 采用超流水线结构，取指令 (IF, IS) 和取数 (DF, DS) 都要跨越两个流水线级，每个时钟周期包含两个流水级，处理器取两条指令都要访问指令 Cache。在寄存器流水级 (RF) 的开始，指令已到了指令寄存器中，可以进行指令译码，并且访问寄存器堆。指令 Cache 采用直接映像方式，因此，从指令 Cache 中读出的区号与访问存储器的物理地址比较，如果相等，表示指令 Cache 命中。对于非存储器操作指令，如果指令 Cache 命中，在指令执行流水级 (EX) 执行，其结果在 EX 的末尾得到。

4.6.3 超长指令字处理机

超长指令字处理机 (Very Long Instruction Word, VLIW) 是指一条指令内可以包含多个同时执行的操作，因而其指令字特别长。例如，Multiflow VLIW 处理机指令字长度为 256 位或 1024 位，最多允许有 7 条指令并发执行，采用微程序控制实现。VLIW 处理机组成与指令格式如图 4-52 所示。VLIW 处理机并发地使用多个功能部件，所有功能部件共享一个大型寄存器堆，功能部件同时执行的操作由硬件进行同步。一条长指令中不同字段有不同的操作码，它们被分派到不同的功能部件。所以，VLIW 是对水平微代码编码的扩展。

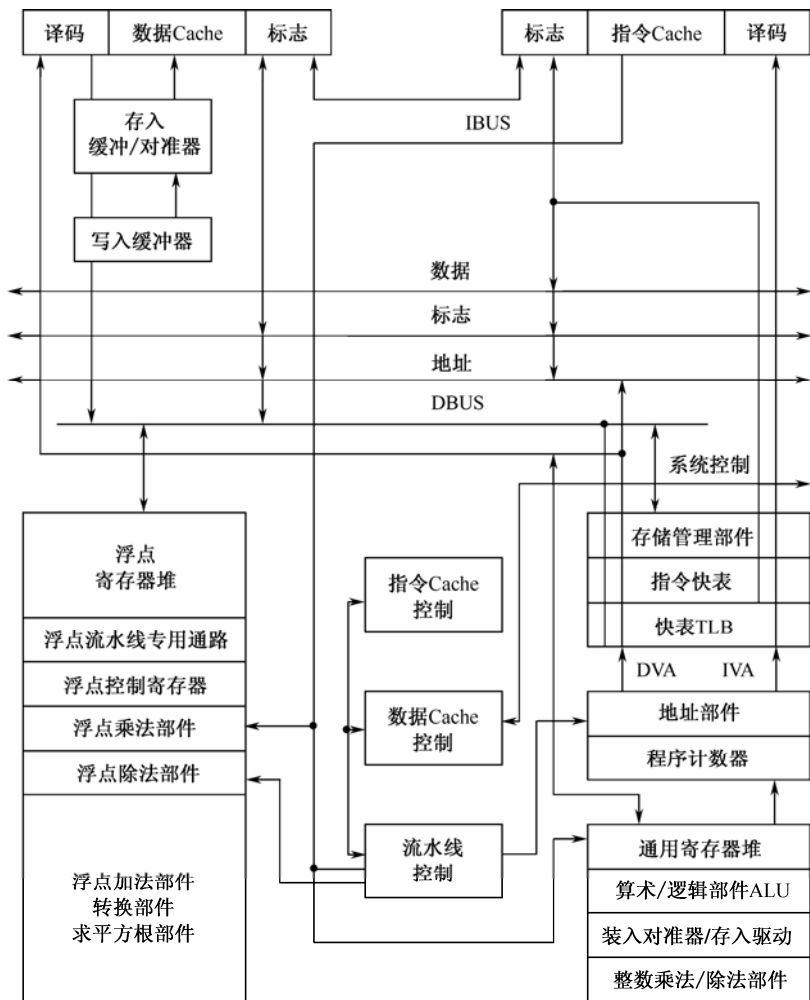


图 4-50 MIPS R4000 超流水线处理机结构

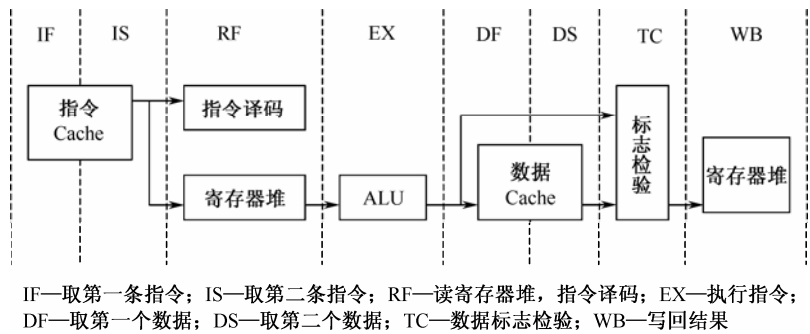


图 4-51 MIPS R4000 处理机的流水线操作

在 VLIW 结构中，指令并行性和数据传送完全是在编译时确定的，运行时的资源调度和同步则被完全排除，因此 VLIW 是超量量的一个极端特例，在 VLIW 中所有独立或不相关的操作在编译时已打包在一起。用普通的 RISC 指令字（长度为 32 位）书写的程序必须压缩成一条 VLIW 指令，代码压缩工作必须由编译器完成。编译器要具有能跟踪程序流并利用跟踪

信息预测转移方向的能力。

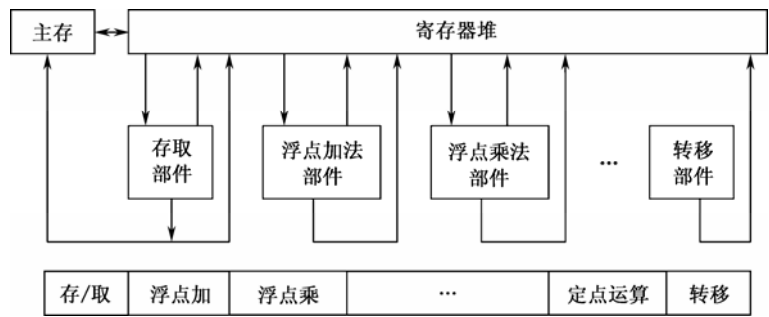


图 4-52 VLIW 处理机组成与指令格式

VLIW 处理机执行指令流水线时空图如图 4-53 所示，每条指令能指定多个操作。本例中 CPI 为 0.33。VLIW 机与超标量机区别如下：

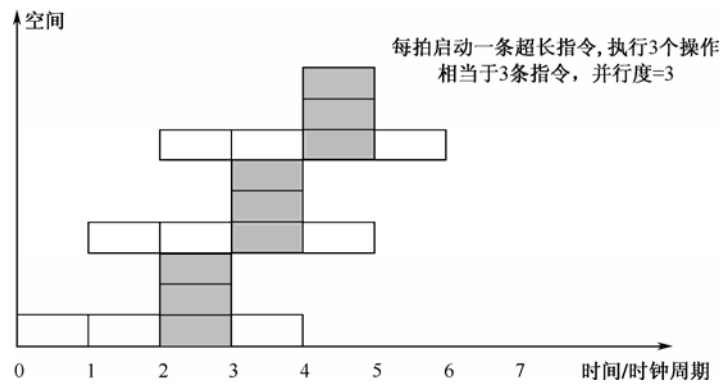


图 4-53 超长指令字（VLIW）的指令流水线时空图

- (1) VLIW 指令译码比超标量指令更容易。
- (2) 当超标量机可用的指令级并行性比由 VLIW 机可开发的相应值小时，超标量机的代码密度更为紧凑。
- (3) 超标量机可以和非并行机系列的目标代码兼容，而 VLIW 机开发的并行性不同时，需要不同的指令系统。

VLIW 机优点如下：

- (1) VLIW 指令中并行操作同步在编译时完成，从而提高了处理机效率。
- (2) 当用户代码中有高的指令级并行性（Instruction Level Parallelism, ILP）时，VLIW 程序代码长度要短得多，因此经编译后的 VLIW 目标程序执行时间短。
- (3) 大大简化运行时的资源调度，因为 VLIW 体系结构中指令并行性和数据移动是在编译时解决的。

VLIW 机缺点如下：

- (1) 必须要有智能编译器的支持。在 VLIW 体系结构中开发的是标量运算中的随机并行性，与向量机中开发的有规则并行性及 SIMD 计算机中锁步并行性不同。没有智能编译器有效代码生成，就无法使用 VLIW 机。

(2) 软件兼容性差。必须解决与其他系列机软件向后兼容性的问题。

(3) 软件的可移植性差。为一台 VLIW 机编译的软件必须重新编译才能在另一台 VLIW 机上使用。当不同代的 VLIW 机的 ILP 增加时，必须开发新的智能编译器。

表 4-4 列出了 VLIW 与超标量体系结构的区别。

表 4-4 VLIW 与超标量体系结构的性能比较

性能 体系结构	硬件支持	并行性开发时间	代码密度	平均 CPI	兼容性	可移植性
VLIW	简单	编译时	差	低	无	无
超标量	复杂	运行时	好	高	有	有

VLIW 技术是以硬件最容易执行的方式排列指令的，在编译时开发其并行性，而超标量技术是以硬件做并行处理，若想在速度上有所突破，VLIW 是一种重要的选择。Intel 和 HP 公司联合开发的基于 IA—64 结构的处理器考虑采用 VLIW 以开发更多的并行性。美国 Transmeta 公司推出的 Crusoe 处理器是 VLIW 架构，主频为 700MHz 的 TM5400，采用 128 位的 VLIW 核心，可同时接收 4 条指令，用 0.18μm 工艺在 73mm² 面积上集成了 2370 万个晶体管。2003 年发布的 TM6000 系列处理器主频达到 1GHz 以上。最新的 TM8000 处理器采用 256 位 VLIW 核心，主频超过 1GHz，耗电 0.5W，同时可接收 8 条指令。

Intel 基于 IA—64 结构的 Itanium（安腾）处理器和 AMD 基于 x86—64 结构的 Athlon（速龙）处理器也采用 VLIW 技术来提高处理器性能。Intel 的 IA—64 是寄存器型的 RISC 指令集。IA—64 寄存器包括：128 个 64 位通用寄存器，实际上是 65 位；128 个 82 位的浮点寄存器，比标准的 80 位 IEEE 格式多 2 位标志位；64 个 1 位的条件寄存器；8 个 64 位分支寄存器，用于间接寻址分支指令；若干用于系统控制、内存映射、性能计数器以及与操作系统通信的寄存器。整数寄存器采用寄存器堆栈式机制，加速过程调用。IA—64 提供特殊的 Load 和 Store 指令来保存和恢复寄存器堆栈，用特殊硬件（寄存器堆栈引擎）处理寄存器堆栈溢出。除了整数寄存器，还有浮点寄存器（用于保存浮点数据）、分支寄存器（用于保存间接分支指令的目标地址）、条件寄存器（用于保存条件控制指令的条件位）。IA—64 系统结构得到 VLIW 大部分支持。同一条指令中操作具有隐式并行性，拥有固定格式的操作字段，比 VLIW 拥有更大的灵活性。一方面通过编译器检测指令级并行性，并把指令调度到并行的指令槽中；另一方面在指令格式上增加灵活性，由编译器标识出那些无法与后续指令并行执行的指令。为了得到隐式并行并简化指令译码，IA—64 把指令安放在指令组内，而指令的固定格式则通过包含三条指令的集束（Bunble）实现。集束宽度为 128 位，其中有 5 位模板字段和三条指令（每条指令长度位 41 位）。为了简化译码和指令发射过程，集束的模板字段指明模板中每条指令需要什么类型的执行单元。

IA—64 是 HP 和 Intel 公司于 1995 年开始合作设计的一种新的系统结构，以取代 x86（IA—32）。Itanium（安腾）处理器是 IA—64 系统结构的第一个实现样本，发布于 2001 年年中，主振 800MHz。Itanium 每个时钟周期可以发射 6 条指令，其中最多三条分支指令和两条访内指令。有三级 Cache，第一级是指令和 data 分开的 Cache，只放定点数，不放浮点数；第二级和第三级是一体 Cache。第三级是 4MB 片外 Cache。Itanium 有 9 个功能部件：2 个整数部件、2 个访内部件、3 个分支部件和 2 个浮点部件，且所有功能部件都是流水线化的。Itanium 的

指令发射窗口可以同时包含两个集束，在一个周期内最多发射 6 条指令。Itanium 根据集束模板位把指令分配到功能部件中，忽略空操作指令（NOP）和条件不成立的条件执行指令（条件转移指令和条件调用指令）。如果待发射的指令所需的功能部件已被占用而导致暂停发射，此时需要分割集束，被分割集束仍然占用一个集束位置，是一条待执行的指令，而第二个集束是三条指令。所以，最坏情况下，集束分割发射，硬件只能得到 4 条指令。

Itanium 采用 10 级流水线，分 4 个部分：

（1）前端（IPG, Fetch, Rotate 阶段）。每个时钟周期预取 2 个集束（共 32B）到预取缓存，即预取 6 条指令，缓存容量为 128B，可预取 8 个集束（24 条指令），采用多级自适应预测器进行分支预测。

（2）指令传送（EXP, REN 阶段）。把最多 6 条指令分配到 9 个功能部件中，为旋转（Rotate）寄存器和堆栈寄存器重命名。

（3）操作数传送（WLD, REG 阶段）。访问寄存器堆，执行寄存器直接通路，访问和更新寄存器记分板，检查相关的条件位。记分板用来检测什么时候指令可以继续执行，当集束里一条指令停顿不会导致整个集束被停顿。

（4）执行（EXE, DET, WRB 阶段）。通过 ALU 和 Load/Store 部件执行指令，检测异常；设置 NaT 位（Not a Thing），即猜测指令异常延迟处理位，用于整数寄存器；浮点寄存器由 NaTVal（Not a Thing Value）位实现。本阶段结束作废指令，执行回写。

在嵌入式领域应用 VLIW 概念的典型芯片是 Trimedia 和 Crusoe。Trimedia 采用压缩指令机制，即指令在主存和 Cache 内是压缩的，而在取指时解压。这种方法克服了 VLIW 代码占用空间大的缺点，代码过大不适合嵌入式领域。Crusoe 是专为低功耗、移动应用而设计的 VLIW 处理器，其特点是通过软件把 x86 指令集转换成 VLIW 指令集，从而实现 x86 指令集和 VLIW 指令集兼容。

4.6.4 超标量超流水 VLIW 处理机

超标量处理机主要开发空间并行性，依靠多个操作在重复设置的操作部件上同时执行来提高程序的执行速度。在一个时钟周期内同时发射多条指令，同时执行并完成多条指令。超流水处理机主要开发时间并行性，把一个功能段细分为几个流水级，或者说把一个时钟周期细分为多个流水线周期，每个流水线周期发射一条指令，则一个时钟周期能够发射并执行多条指令，从而在一个操作部件上重叠多个操作，通过较快时钟周期提高程序的执行速度。VLIW 处理机使用多个独立操作部件，由编译器开发程序中隐性并行性，将多个操作组合成一条长指令，实现由一条指令同时完成若干个不相关的操作，达到提高程序执行速度的目的。超标量超流水 VLIW 处理机为三者的结合，充分利用三者优点，使流水线达到更高的速度。如指令流水处理过程有 4 个阶段：取指（IF）、指令译码（ID）、执行（EX）和写回（WB），则指令在超标量超流水 VLIW 流水线的操作情况（时空图）如图 4-54 所示。每个时钟周期内发射三次，每次发射三条指令，每条指令有三个操作。设每个功能段延迟时间都是 Δt ，则流水线从启动到满负荷运行，完成 36 个任务只要 $5\Delta t$ ，完成 72 个任务只要 $6.33\Delta t$ 的时间。与非流水线顺序执行单处理机相比，在主振相同前提下，机器运算速度在理论上可提高 108 倍。

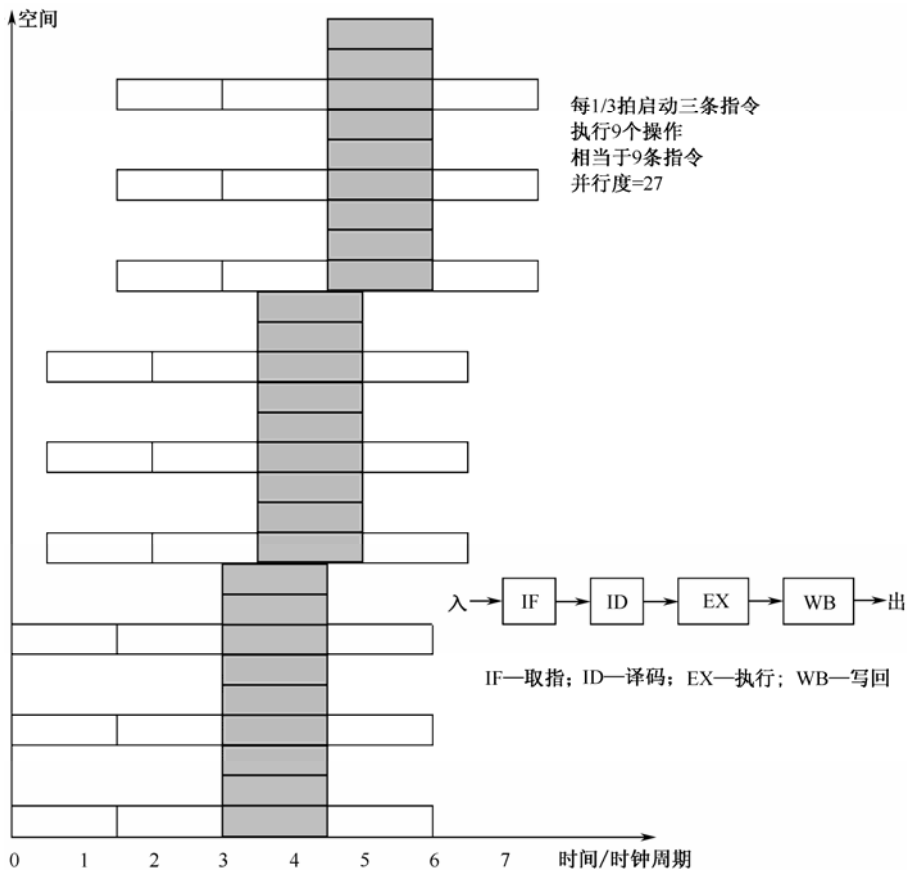


图 4-54 三发射超标量超流水 VLIW 流水线时空图

设流水线为 K 级 (即 K 个功能段), 每级执行时间均为 Δt , 超流水处理器时钟频率为主机时钟平均的 n 倍, 超标量处理器发射度为 m , VLIW 处理器每条指令实现 r 个操作, 若执行 N 条指令, 则对于以下 8 种情况的执行时间分别描述如下。

- (1) 单发射基准流水线 $T(1, 1, 1)=K \cdot \Delta t+(N-1) \cdot \Delta t$
- (2) m 发射超标量流水线 $T(m, 1, 1)=K \cdot \Delta t+\left\lceil \frac{N-m}{m} \right\rceil \cdot \Delta t$
- (3) 单发射 n 倍超流水线 $T(1, n, 1)=K \cdot \Delta t+(N-1) \cdot \frac{\Delta t}{n}$
- (4) 单发射 r 操作 VLIW 流水线 $T(1, 1, r)=K \cdot \Delta t+\left\lceil \frac{N-r}{r} \right\rceil \cdot \Delta t$
- (5) m 发射 n 倍超标量流水线 $T(m, n, 1)=K \cdot \Delta t+\left\lceil \frac{N-m}{m} \right\rceil \cdot \frac{\Delta t}{n}$
- (6) m 发射 r 操作超标量 VLIW 流水线 $T(m, 1, r)=K \cdot \Delta t+\left\lceil \frac{N-m \square r}{mr} \right\rceil \cdot \Delta t$
- (7) n 倍 r 操作超流水 VLIW 流水线 $T(1, n, r)=K \cdot \Delta t+\left\lceil \frac{N-r}{r} \right\rceil \cdot \frac{\Delta t}{n}$

(8) m 发射 n 倍 r 操作超标量超流水 VLIW 流水线

$$T(m, n, r) = K \cdot \Delta t + \left\lceil \frac{N - m \cdot r}{mr} \right\rceil \cdot \frac{\Delta t}{n}$$

m 发射超标量机的速度最多为单发射机的 m 倍； n 倍时钟速率的超流水机的速度最多为单发射机的 n 倍；一条指令执行 r 个操作的 VLIW 机的速度最多为单发射机的 r 倍。三者结合的超标量超流水 VLIW 机在理论上的速度最多是单发射机的 mnr 倍，是非流水线顺序执行的计算机的 $kmnr$ 倍。

例如，设现有 150 个任务需要进入流水线，已知流水线功能段都为 8 个，流经每个功能段的时间相同，都是 $\Delta t = 10\text{ns}$ ，现计算下列情况下完成 150 个任务分别需要多少时间。

解： $K=8$ ， $\Delta t=10\text{ns}$ ， $N=150$ 。

(1) 单发射基准流水线 $T(1, 1, 1) = 8 \times 10 + (150 - 1) \times 10 = 1570\text{ns}$

(2) 超标量流水线每个时钟周期发射 4 条指令， $m=4$ 。

$$T(4, 1, 1) = 8 \times 10 + \left\lceil \frac{150 - 4}{4} \right\rceil \times 10 = 450\text{ns}$$

(3) 超流水线，每个时钟周期可以分时发射 4 次，每次发射 1 条指令， $n=4$ 。

$$T(1, 4, 1) = 8 \times 10 + (150 - 1) \times \frac{10}{4} = 452.5\text{ns}$$

(4) VLIW 机，每个时钟周期发射 1 条指令，每条指令可执行 3 个不相关操作， $r=3$ 。

$$T(1, 1, 3) = 8 \times 10 + \left\lceil \frac{150 - 3}{3} \right\rceil \times 10 = 570\text{ns}$$

(5) 超标量超流水线，每个时钟周期可以分时发射 4 次，每次发射 4 条指令， $m=4$ ， $n=4$ 。

$$T(4, 4, 1) = 8 \times 10 + \left\lceil \frac{150 - 4 \times 4}{4 \times 4} \right\rceil \times 10 = 170\text{ns}$$

(6) 超标量 VLIW 机，每个时钟周期发射 4 条指令，每条指令可执行 3 个不相关操作， $m=4$ ， $r=3$ 。

$$T(4, 1, 3) = 8 \times 10 + \left\lceil \frac{150 - 4 \times 3}{4 \times 3} \right\rceil \times 10 = 200\text{ns}$$

(7) 超流水 VLIW 机，每个时钟周期可以分时发射 4 条指令，每次发射 1 条指令，可执行 3 个不相关操作， $n=4$ ， $r=3$ 。

$$T(1, 4, 3) = 8 \times 10 + \left\lceil \frac{150 - 4 \times 3}{4 \times 3} \right\rceil \times 10 = 200\text{ns}$$

(8) 超标量超流水 VLIW 机，每个时钟周期可以分时发射 4 条指令，每次发射 4 条指令，每条指令可执行 3 个不相关操作， $m=4$ ， $n=4$ ， $r=3$ 。

$$T(4, 4, 3) = 8 \times 10 + \left\lceil \frac{150 - 4 \times 3}{4 \times 3} \right\rceil \times \frac{10}{4} = 110\text{ns}$$

超标量超流水 VLIW 机相对于单发射基准流水线的加速比为

$$S(m, n, r) = \frac{T(1,1,1)}{T(m,n,r)} = \frac{mnr(K + N - 1)}{Kmnr + N - mr}$$

当指令执行条数 $N \rightarrow \infty$ 时，超标量超流水 VLIW 机的加速比最大值为 $S(m, n, r) = mnr$ 。

$$\text{上例加速比最大值} \quad S(4, 4, 3) = \frac{T(1,1,1)}{T(4,4,3)} = \frac{1570}{110} = 14.3$$

1992年美国DEC公司（已被Compaq公司并购，而Compaq公司已与HP公司合并）推出的Alpha21064处理器采用超标量超流水结构，如图4-55所示。Alpha21064处理器是RISC处理器，字长64位，主振200MHz，采用0.75μm CMOS—4工艺，峰值速度为400MIPS。Alpha21064由4个部件和两个Cache组成。EBOX是整数执行部件；FBOX是浮点执行部件；ABOX是地址部件；IBOX是中央控制部件。IBOX负责取指、指令译码、指令发射、流水线控制、程序计数器PC计算等。IBOX可以同时从指令Cache读入两条指令并进行译码，而且对这两条指令进行资源冲突检测、数据相关和控制相关分析。如资源和相关性允许，IBOX就把两条指令同时发射给EBOX，ABOX和FBOX。EBOX内有一个由32个64位寄存器组成的定点寄存器堆，有4个读出端口和两个写入端口，可以把两个源操作数或结果送EBOX及ABOX。操作部件数据宽度为64位，有多条专用数据通路，可以把运算结果直接送执行部件。FBOX采用流水线结构，有两套浮点操作指令，一套是DEC格式的，另一套是IEEE 754格式的，共有36条浮点操作指令。FBOX内32×64位的浮点寄存器堆有三个输出端口和两个输入端口。它还有一个用户可访问的控制寄存器FPCR，内有舍入控制、陷阱允许、异常事故标志等。除了除法指令外，FBOX每个流水线周期可以接收一条指令，指令延迟时间是6个流水线周期。FBOX也有专用数据通路，当发生数据相关时，把结果通过专用数据通路直接送执行部件。ABOX包括地址发生器、存储管理部件（数据快表）、读数缓冲栈和写数缓冲栈。Alpha21064芯片内有两个Cache：数据Cache和指令Cache，容量都为8KB，采用直接映像方式，逻辑地址中包含一个区号字段。另外，由于采用动态转移预测技术，指令Cache中，每个数据块（32B）包含一个8位转移历史字段。Alpha21064共有三条指令流水线，每个流水线周期可以发射两条指令，整数操作和地址计算分为7个功能段，浮点操作为10个功能段，三条指令流水线平均段数为8段。按照一般定义，每个时钟周期（即流水线周期）发射多条指令的处理机为超标量机，指令流水线级数级大于等于8级为超流水线机，所以Alpha21064应该是超标量超流水线处理机。

DEC公司在1998年推出的Alpha21264采用0.35μm CMOS—6工艺，集成1520万个晶体管，有4条整数流水线和2条浮点流水线，每个时钟周期可执行4条整数指令，最多可执行6条指令，Alpha21264也属于超标量超流水线机。

在一条指令级并行度为 (m, n) 的超标量超流水线处理机上，连续执行 N 条无资源冲突、无数据相关和控制相关的指令所需时间为

$$T(m, n) = (K' + \frac{N-m}{mn}) \cdot \Delta t$$

式中， K' 是指令流水线的时钟周期数，而不是流水线级数。例如Alpha21064的 $K'=4$ 。 Δt 是一个时钟周期的时间长度。式中第一项是流水开始建立时 m 条指令通过流水线所需要的时间，而第二项是执行其余 $(N-m)$ 条指令所需要的时间，这时每个时钟周期平均执行 mn 条指令。

单流水线处理机连续执行 N 条指令所需时间为

$$T(1, 1) = (K + N - 1) \cdot \Delta t$$

式中， K 是流水线的级数。因此，超标量超流水线处理机相对于单流水线处理机的加速比为

$$S(m, n) = \frac{T(1, 1)}{T(m, n)} = \frac{mn(K + N - 1)}{mnK' + N - m}$$

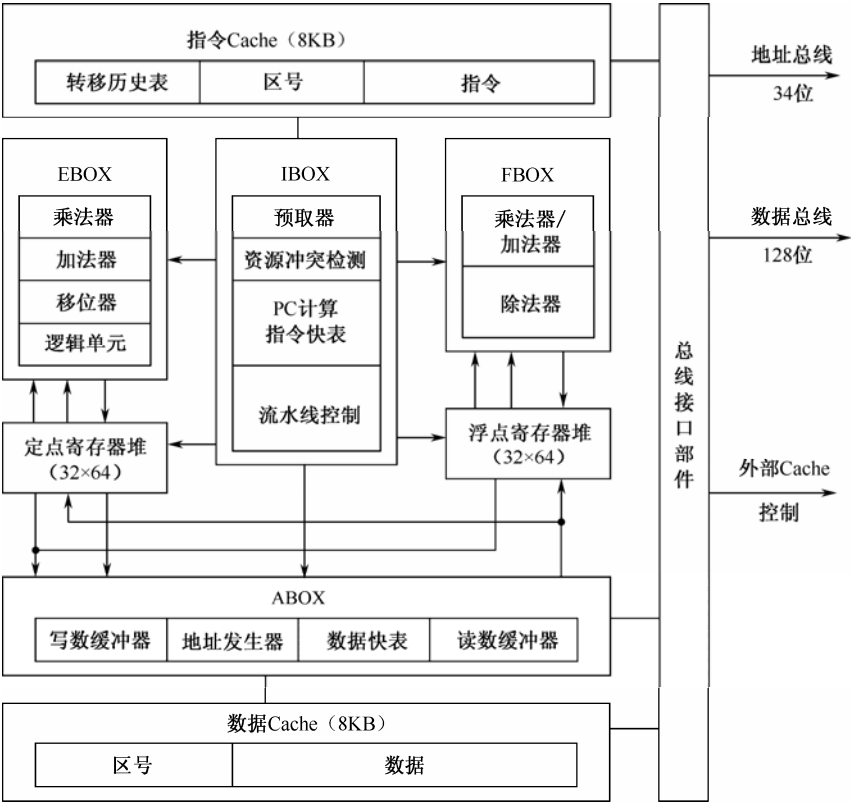


图 4-55 Alpha21064 处理机结构

当指令执行的条数 $N \rightarrow \infty$ 时，在理想情况下，加速比的最大值为

$$S(m, n) = mn$$

图 4-56 给出了超标量处理机、超流水线处理机和超标量超流水线处理机相对于单流水线处理机的性能曲线。图中横坐标是指令级并行度，用 mn 表示，纵坐标为相对性能，也可理解为实际的加速比，或者实际指令级并行度。因此，也可将图 4-56 曲线的横坐标理解成设计指令级并行度，或最大指令级并行度，而纵坐标表示能达到的实际指令级并行度。

从图 4-56 可得到如下结论：

(1) 超标量处理机的相对性能最高，主要原因如下：

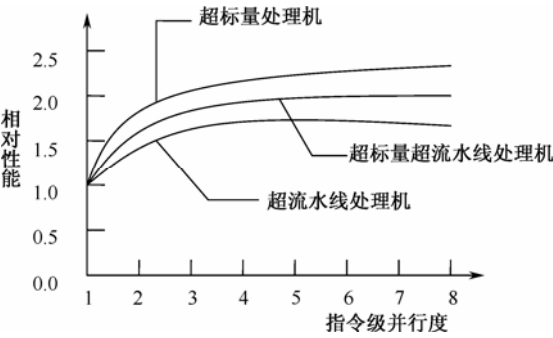


图 4-56 三种指令级并行处理机的相对性能比较

① 超标量处理机在每个时钟周期的开始就同时发射多条指令，而超流水线处理机要把一个时钟周期平均分成多个流水线周期，每个流水线周期发射一条指令，因此启动时间比较长。

② 超流水线处理机条件转移造成的损失比超标量处理机大。

③ 对指令执行过程中的功能段，超标量处理机采用重复设置相同的执行部件，而超流水线处理机只是把同一个执行部件细分为多个流水级。因此，超标量处理机指令执行部件的冲突比超流水线处理机要小。

(2) 从图 4-56 看出，设计指令级并行度较低时，实际指令级并行度提高较快，但是随着设计指令级并行度增加，实际指令级并行度提升越来越慢。因此实际设计时，一般认为 $m \leq 4$, $n \leq 4$ 。

(3) 对于一个特定程序，由于受到本身的数据相关和控制相关的限制，它的指令级并行度的最大值是确定的，由程序自身的语义决定，与程序运行在哪一种处理机上无关。因此，图 4-56 中三条曲线，对于某个特定程序，最终都要收拢到同一个点上。对于不同的程序，收拢点的位置是不同的。由此可见，对于特定程序而言，不断提高设计指令级并行度到某一个程度，采用图中三种超级流水线结构中任一种，其实际效果都是差不多的。

(4) 提高实际指令级并行度还与采用的调度算法有关，要充分开发程序中指令级并行度，实现最优调度非常复杂，通常需要编译器和硬件结合，才能得到较好调度效果。

4.6.5 P6 微结构

Intel P6 微系统结构是 Pentium Pro, Pentium II 和 Pentium III 的基础，其专用的扩展指令集（MMX 和 SSE）相同，而在时钟频率、Cache 结构和内存接口上都有不同之处，如表 4-5 所示。

表 4-5 基于 P6 微系统结构的 Intel 处理器及其主要区别

处 理 器	首次发布日期	时钟频率范围	一级 Cache	二级 Cache
Pentium Pro	1995	100~200MHz	8KB 指令+8KB 数据	256 KB ~1024KB
Pentium II	1998	233~450MHz	16KB 指令+16KB 数据	256 KB ~512KB
Pentium II Xeon	1999	400~450MHz	16KB 指令+16KB 数据	512KB ~2MB
Celerom	1999	500~900MHz	16KB 指令+16KB 数据	128 KB
Pentium III	1999	450~1100MHz	16KB 指令+16KB 数据	256 KB ~512KB
Pentium III Xeon	2000	700~900MHz	16KB 指令+16KB 数据	1MB ~2MB

Pentium II 增加了 MMX 扩展指令，而 Pentium III 增加了 SSE 扩展指令。在 Pentium Pro 中，处理器和 Cache 整合在一个多芯片的模块中。Pentium II 使用标准 Cache。Pentium III 不但有片内 256KB 二级 Cache，还有片外的 512KB Cache。Xeon 处理器主要面向服务器应用，使用片外二级 Cache，并支持多处理器结构。

P6 微系统结构是动态调度的处理器，把每条 IA—32 指令翻译成流水线执行的微操作，微操作类似于标准 RISC 指令。在一个时钟周期内，最多可取出 3 条 IA—32 指令，并译码为微操作。每个时钟周期内最多产生 6 个微操作，其中 4 个分配给第一条 IA—32 指令，另外两个分别分配给其余两条 IA—32 指令。流水线由 14 个流水段组成：8 个流水段用于有序的取指令、译码和发射；三个流水段用于功能部件中的乱序执行，共有 5 个功能部件（定点部

表 4-6 P6 微系统结构常见的重复率

指令名称	流水线段数	重复率
整数 ALU	1	1
整数 Load	3	1
整数乘	4	1
浮点加	3	1
浮点乘	5	2
浮点除（64 位）	32	32

件、浮点部件、分支部件、内存地址部件和内存访问部件)；三个流水段用于指令提交。表 4-6 是 P6 微系统结构常见的重复率，重复率等于 1 表示那个部件是完全流水线化的，重复率等于 2 表示每隔一个周期可以开始一个新的操作。

图 4-57 反映了各流水段的吞吐量、流水段之间的缓存关系。如某个流水段的输入缓存不能提供足够的操作数或者输出缓存已满，则该流水段就达不到最大吞吐量。其次，在所有功能部件中，内部限制和动态事件（如 Cache 不命中）发生都会导致流水线停顿。例如指令 Cache 不命中使取指流水段得不到三条 IA—32 指令（即 16B 的指令）；三条指令在译码过程中要受到它们与微指令之间的映射关系的限制。

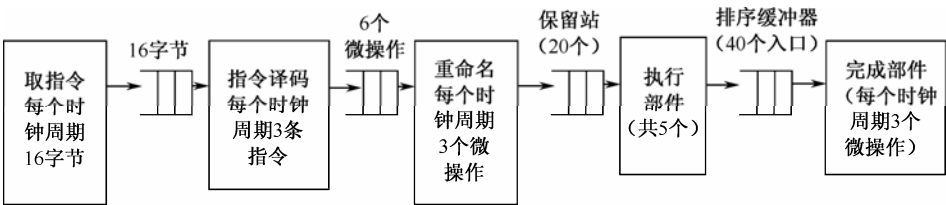


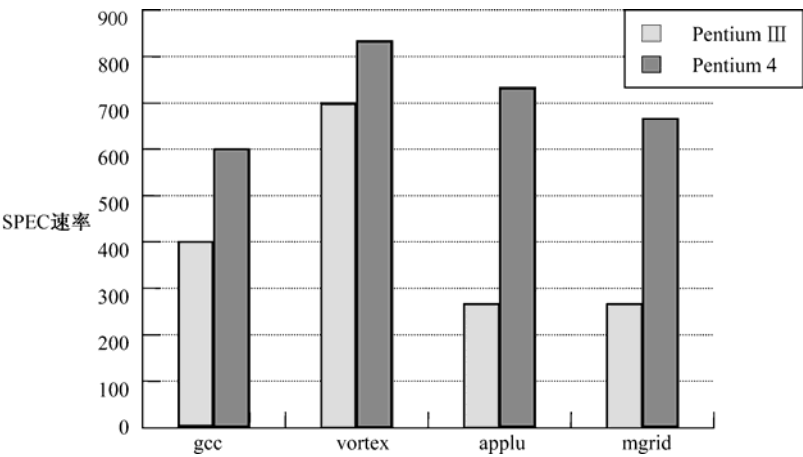
图 4-57 P6 处理器流水线中每个流水段的吞吐量和各个流水段之间的缓存关系

Pentium 4 的微系统结构称为 NetBurst，与 Pentium III 的结构类似，都是每个时钟周期最多可取三条指令，并译码成相应的微操作，发送到一个乱序执行引擎上，这个执行引擎（即执行部件）每个周期最多能够执行三个微操作。Pentium 4 与 Pentium III 区别也很多，主要不同如下：

- NetBurst 有更深的流水线。一条加法指令，P6 使用 10 个时钟周期，而 NetBurst 使用 20 个时钟周期，其中包括 2 个专门用于传递运算结果的时钟周期。
 - NetBurst 使用寄存器重命名技术（类似 MIPS R10000 和 Alpha 21264），P6 中使用排序缓冲器，采用寄存器重命名技术可以保存 128 个中间结果，而 P6 只能保存 40 个。
 - NetBurst 有 7 个定点执行部件，P6 只有 5 个，增加了专门的定点 ALU 和专门的地址计算部件。
 - NetBurst 的 ALU 以两倍时钟速度运行，配上强大的数据 Cache，使 ALU 基本操作延迟 0.5 个周期（P6 延迟 1 个周期），写入操作延迟 2 个周期（P6 延迟 3 个周期）。
 - NetBurst 使用复杂的 Cache 调度技术来缩短取指时间，而 P6 用一般的指令缓冲器和指令 Cache。
 - NetBurst 有一个很大的分支目标缓存，比 P6 大 8 倍，并且改进了预测算法。
 - NetBurst 一级 Cache 为 8KB，P6 则为 16KB，但 NetBurst 的二级 Cache（256KB）容量更大，频带更宽。
 - NetBurst 有新的 SSE2 浮点指令，每周周期启动两个浮点操作，两个操作被组织成 128 位的 SIMD 或者短向量结构，因此浮点运算能力有很大提高。
- 总而言之，NetBurst 结构可以在更高的时钟频率上工作，并且能够以接近峰值的速度持

续运行。

图 4-58 表示采用 SPEC95 和 SPEC2000 测试程序（两个整数 gcc 和 vortex，两个浮点数 applu 和 mgrid）对 Pentium III 和 Pentium 4 进行测试，Pentium 4 的性能大概是 Pentium III 的 1.2~2.9 倍，其中浮点测试程序明显发挥了 Pentium 4 中新的扩展指令集（SSE2）的作用。浮点性能的提高超过时钟频率增幅 1.6~1.7 倍。



（两个整数 gcc 和 vortex，两个浮点数 applu 和 mgrid）

图 4-58 4 个 SPEC2000 基准测试程序测试结果比较图

第5章 并行处理机

计算机系统设计的基本任务之一就是加快指令的解释过程。除了采用高速部件，努力提高指令内部的并行性，从而加快单条指令的解释过程之外，也可以采用提高指令间的并行性，从而并发地解释两条或两条以上的指令，使整段程序达到提高计算机整体速度的目的。本章在叙述并行性基本概念的基础上，着重介绍并行处理机、互连网络、阵列处理机、相联处理机等基本原理、基本结构及其性能分析，以及设计中应考虑的一些问题。

5.1 系统结构中的并行性概念

5.1.1 并行性概念

计算机系统增加并行性可以提高机器的处理速度。例如，运算器结构中的串行运算（即 n 位数据用一位运算器）到并行运算（即用 n 位运算器），在元器件速度相同的情况下，后者的运算速度几乎提高到前者的 n 倍。但是，并行性并不限于设备的简单重复，在单处理机系统内采用指令先行提取技术、流水线技术以及并发执行多道程序等均可体现并行性。

因此，并行性（Parallelism）定义为：在同一时刻或同一时间间隔内完成两种或两种以上的性质相同或不同的工作，只要在时间上相互重叠，均存在并行性。并行性又可分为同时性（Simultaneity）和并发性（Concurrency）。所谓同时性是指两个或多个事件在同一时刻发生的并行性；而并发性是指两个或多个事件在同一时间间隔内发生的并行性。以 n 位串行进位并行加法为例，由于存在进位信号从低位到高位逐位递进的延迟时间，因此 n 位全加器的运算结果并不是在同一时刻获得，故不存在同时性，只存在并发性。如果有 m 个存储器模块能同时进行存、取信息，则属于同时性。

并行性有不同的等级。从执行角度看，并行性等级可从低到高划分为：

- （1）指令内部并行，即指令内部的微操作之间的并行。
- （2）指令间并行，即并行执行两条或多条指令。
- （3）任务级或过程级并行，即并行执行两个或多个过程或任务（程序段）。
- （4）作业或程序级并行，即在多个作业或程序间并行。

在单处理机系统中，这种并行性升到某一级别后（如任务、作业级并行），需通过软件来实现，如体现在操作系统的进程管理、作业管理中。而在多处理机系统中，由于任务或作业是分配给各个处理机完成的，因此其并行性由硬件实现。可见，实现并行性也有一个软、硬件功能合理分配的问题，往往需要全面综合考虑。

从处理数据的角度看，并行性等级从低到高可分为：

- （1）字串位串，同时只对一个字的一位进行处理，不存在并行性。
- （2）字串位并，同时对一个字的所有位进行处理，已开始出现并行性。
- （3）字并位串，同时对多个字的同一位（如位片机）进行处理。
- （4）字并位并（全并行），对多个字的所有位或部分位进行同时处理，这是最高一级的

并行。

计算机系统提高并行性的措施很多，就其思想而言，可归纳为下列三种技术途径。

1. 时间重叠 (Time-Interleaving)

在并行性概念中引入时间因素，即多个处理过程在时间上互相错开，轮流重叠使用同一套硬件的各个部件，以加快部件的周转而提高速度。指令的重叠解释是最简单的时间重叠。时间重叠原则上不要求重复的硬件设备，能保证计算机系统具有较高的性能价格比。

2. 资源重复 (Resource-Replication)

在并行性概念中引入空间因素，根据“数量取胜”原则，重复设置硬件资源以大幅度提高计算机系统的性能。过去，由于价格原因，不能普遍采用。随着硬件价格不断下降，从单处理机发展到多处理机，资源重复已经成为提高系统并行性的有效手段。

3. 资源共享 (Resource-Sharing)

利用软件方法，使多个用户分时使用同一个计算机系统。例如多道程序、分时系统就是资源共享的产物。它是提高计算机系统资源利用率的有效措施。

在一个计算机系统内，可以通过多重技术途径，采用多种并行措施，既有执行程序的并行性，又有处理数据的并行性。例如，如果执行程序的并行性达到任务或过程级，处理数据的并行性达到字并位串级，即可认为该系统已经进入并行处理领域。所以，并行处理 (Parallel Processing) 是信息处理的一种有效形式。通过开发计算过程中的并行事件，可使并行性达到较高的级别。并行处理是硬件、软件、算法、语言等多方面综合研究的领域。

5.1.2 并行处理的发展

计算机系统并行处理的发展过程如图 5-1 所示。

1. 单机系统中并行处理的发展

单处理机并行性开发主要是时间重叠这个途径。实现时间重叠的基础是部件功能专用化 (Functional Specialization)。将一件工作按功能分割成若干联系的部分，每一部分有指定的专门部件来完成，然后按时间重叠的原则把各部分执行过程在时间上重叠起来，使所有部件依次分工完成一组同样工作，在处理机内部能同时解释两条指令，从而提高处理机速度，这种方式称为重叠方式。如果把指令解释过程分解成多个子过程，分别由多个专用部件完成，这就是先行控制，可实现同时解释多条指令。这些处理机实现了指令间并行，如把功能专用化深入到处理机的执行部件内部，将该部分再分成多个专用功能段，进行流水处理，从而对数据处理的并行性由字串位并发展到全并行，这就是操作流水线。如果并行的指令间无任何联系，原来用多条指令对向量各元素分别处理，现在可以用一条向量指令对向量的各个元素进行流水处理，这就可以实现从指令间并行性转为指令内的并行性，在一条向量指令内对数据处理达到了全并行。在指令内并行的基础上，增加指令间并行性，这就是向量处理机。向量机的出现表征单处理机进入了并行处理领域。把时间重叠原理应用于任务一级，对各任务设置专用处理机，按流水线方式工作，就构成了宏流水线，即进入了多处理机领域。这种多处理机称为非对称型 (Asymmetrical) 或异构型多处理机 (Heterogeneous Multiprocessor System)。它们由多个不同类型担负不同功能的处理机构成，按作业要求次序，利用时间重叠原理，依次执行多个任务，各自实现规定的操作。

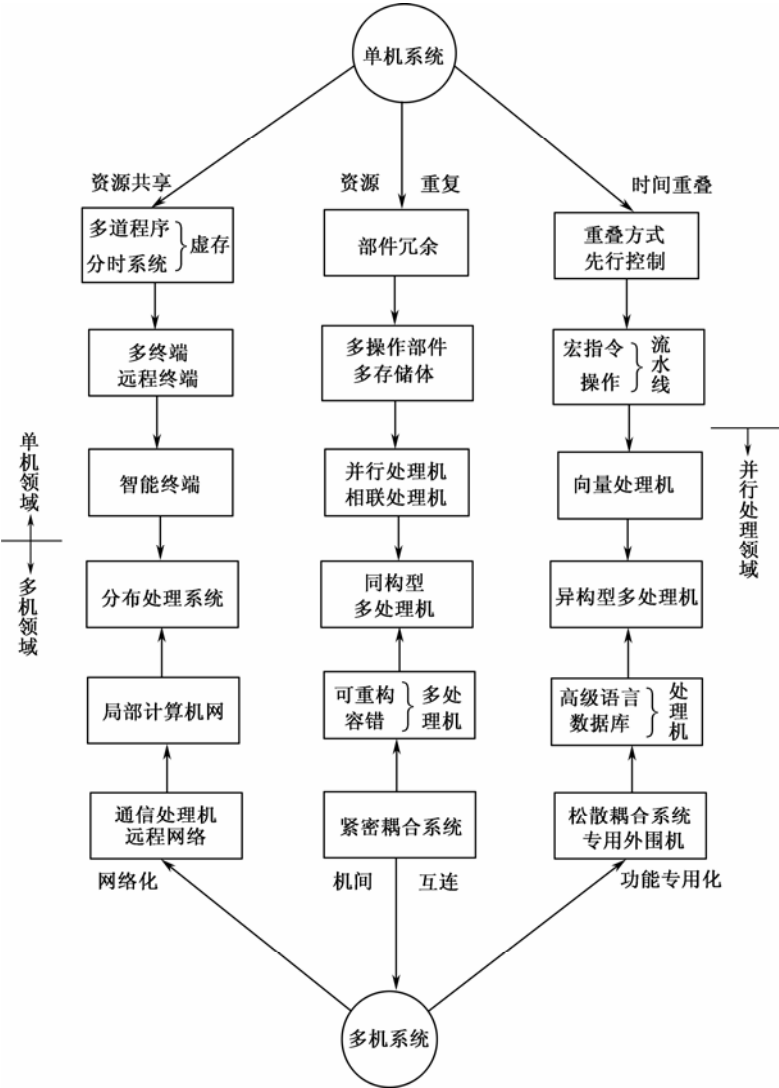


图 5-1 计算机系统并行处理的发展过程

在高性能单机系统中，随着硬件价格下降，资源重复也逐渐普遍起来。资源重复主要是为了提高系统的可靠性，即在关键部件上采用冗余技术。同时，也是为了提高系统的速度。不论处理机是否采用流水线技术，在系统结构中，采用多操作部件和多存储体，都是资源重复的结构。当一条指令正在某操作部件上执行时，只要下一条指令所需要的操作部件没有被占用，它就可以占用，从而实现指令间并行。进一步发展，可以把原来具有各个专门功能的操作部件演变成通用处理单元，为向量并行处理创造条件。通过重复设置多个相同的处理单元，在一个控制器的指挥下，按照同一指令（即一条向量指令）要求，各处理机同时对各向量元素进行操作，这就是并行处理机。它在指令内部实现了数据处理的全并行，将单机系统的并行性上升一级，进入并行处理领域。并行处理机普遍采用阵列结构形式，故称阵列机。

相联存储器是一种按内容寻址的、具有信息处理功能的存储器，能按字并位串或全并行方式对所有存储单元的内容进行操作。以相联存储器为核心，加上中央处理机、指令存储器

和 I/O 接口, 就可以构成以存储器并行动作为特征的相联处理机。它是将并行处理机思想应用于相联存储器内部。显然, 并行处理机和相联处理机还属于单机系统范畴, 虽然数据处理已达到全部并行程度, 但系统中只有一个控制指令部件, 指令的并行性最高只能达到指令间的并发执行。如果进一步提高到任务级并行, 则每个处理单元均配备自己的控制器, 能独立地解释、执行指令而称为一台处理机, 这就进入了多处理机系统范畴。这种多机系统称为对称型 (Symmetrical) 或同构型多处理机系统 (Homogeneous Multiprocessor System)。它们由多个同类型、同功能的处理机构成, 能同时处理同一作业中能并行执行的多个任务。

在单机系统中, 要达到任务或作业级并行, 也可以利用资源共享。从多道程序发展到分时系统, 其实质是单机模拟多机功能。分时系统适用于多终端情况, 对于远程用户, 可配接远程终端。如果在终端内配上微处理器, 使其不仅有 I/O 功能和通信功能, 还具有一定的信息存储、分析、处理能力, 这就成为智能终端。智能终端的出现, 使原来“集中”形态向“分布”形态方向发展。如把智能终端升格为一台计算机, 就进入了多机系统, 这种多机系统称为分布式处理系统 (Distributed Processing System)。将若干台具有独立功能的处理机 (或计算机) 相互连接起来, 在操作系统 (或分布式操作系统) 的控制下, 统一协调地运行, 最少依赖于集中的某个软、硬件资源, 这就是分布式处理系统。分时系统是以“集中”为特征, 分布系统是以“分布”为特征, 表面上两者有巨大的差别, 但其思想基础是一致的。分时系统是以虚拟方法模拟多处理机功能, 而分布系统是用真实处理机代替虚拟处理机。分时系统的并行性是并发性, 而分布系统实现的并行性是同时性。

上述仅从并行角度反映了单机系统的发展趋势和技术途径, 对于具体的计算机系统, 可以综合几种技术途径, 采用多种技术措施, 以实现系统各部分负荷的平衡。

2. 多机系统中并行处理的发展

多机系统包括多计算机系统 (Multi-Computer System) 和多处理机系统 (Multi-Processor System)。前者指由多台独立的计算机构成的系统。后者指由多台处理机构成的系统, 每台处理机有自己的控制器, 可以独立执行程序, 共享公共主存和所有外设。从概念上分析, 多计算机系统和多处理机系统区别明显, 但是实际上多处理机系统除了共享一个公共主存外, 每个处理机带有各自的局部存储器, 其本身已逐渐成为一台完整的计算机。此时, 两者在结构上的区别就不重要了。然而, 在操作系统和并行性方面两者仍有区别。多计算机系统中各个计算机各自有自己的操作系统, 它们之间往往通过通道和通信线路实现通信, 以完整文件或数据集合进行信息传递, 以实现作业和任务级的并行。而多处理机系统则由同一操作系统控制, 由于共享存储器, 各处理机之间不但能以完整的文件或数据集合进行信息传递, 也能以向量或单个数据进行通信, 因此它的并行性可深入到同一任务中指令间并行, 甚至可以在多处理机同时执行多条指令, 对同一数组进行处理, 达到处理数据的全并行。需要指出, 处理机是具有控制器 (即能分析指令) 和算术逻辑部件 (即能执行命令) 的, 因此多处理机系统能够同时分析、执行多条指令。而处理单元仅有算术逻辑部件, 它不能分析指令, 多处理单元的阵列处理机只有一个控制器, 同一时刻只能分析一条指令, 所以它属于单机系统, 而不是多处理机系统。

(1) 多机系统的耦合度。耦合度反映了多机系统中各机器之间物理连接的紧密程度和交互作用能力的强弱。多机系统的耦合度可分为最低耦合、松散耦合、紧密耦合等几类。

① 最低耦合 (Least-Coupled System) 的耦合度很低, 仅通过中间存储介质互相通信,

除此之外，各机器间并无物理连接，也无共享的联机硬件资源。例如，独立的外围计算机与主机脱开，只通过磁盘、磁带对主机的输入、输出予以支持，这是最早的双机合作方式。

② 松散耦合（Loosely-Coupled System）也称间接耦合系统（Indirectly-Coupled System），机器之间通过通道或通信线路实现互连，共享某些外围设备（如磁盘、磁带、光盘等）。它的特点是连接频带较低，在文件和数据集合一级进行相互通信。松散耦合系统有两种形式：一种是多台计算机通过通道和共享的外围设备连接，各个机器实现功能上分工（即功能专用化），机器处理结果以文件和数据集合形式送到共享外设，供其他机器调用，从而获得较高的系统使用效率；另一种是计算机网络，各节点计算机通过通信线路连接，在网络操作系统管理下，合理调度软、硬件资源，以求得更大范围内的资源共享，这是当前松散耦合多机系统的典型形式。

③ 紧密耦合（Tightly-Coupled System）也称直接耦合系统（Directly-Coupled System），机器之间通过总线（Bus）或高速开关实现互连，共享主存储器，机器间通信频率高，信息传输速率和吞吐量大，是当前加速并行处理的首选形式。

（2）多机系统的发展。多机系统也沿着时间重叠、资源重复、资源共享的技术途径向前发展，有三种不同的发展方向，在技术措施上与单机系统有所区别，如图 5-1 所示。

在单机系统中为实现时间重叠，设置了多个专用功能部件，而多机系统则将处理功能分解给各个专用处理机完成，实现功能专用化。各处理机之间按照时间重叠原理工作。早期是把一些辅助功能从主机内分离出来。如输入、输出功能分离出来由通道负责处理，由此，通道逐渐发展成 I/O 处理机（IOP）。以后，将其他辅助功能，如系统测试诊断、终端信息预处理等也分离出来，采用松散耦合方式，构成松散耦合的多机系统。进一步发展下去，许多主要功能，如数组运算、高级语言编译、数据库管理等，也逐渐分离出来，分别由专用处理机完成，机器间耦合程度也逐渐加强，发展成异构型多处理机系统。

早期的多机系统并非是为了提高系统的处理速度而开发的，而是为了提高系统的可靠性而设置多台相同类型的计算机，使可靠性从单机系统的部件级冗余提升到处理机一级。各种不同的容错方案，如多数表决、备份等，对多机系统间互连网络的要求是不同的。但正确性和可靠性是最低要求。在提高互连网络的灵活性和可重构性的基础上，发展出可重构系统（Reconfigurable System），这种系统平时几台计算机正常运行，如同多处理机系统一样，当出现故障时，系统就重新组合，以降低规格继续运行，直至故障排除为止。这是将系统容错能力建立在系统结构一级。随着硬件的大幅度降价，现在多处理机系统的并行处理更多地追求的是提高整个系统的处理速度。此时，对机间互连网络的通信速率提出了更高的要求。所以，高传输速率的机间互连网络是实现高速处理的必要条件，也是实现在程序段或任务一级并行工作的必要条件。在此基础上发展出各种紧密耦合系统，以使并行处理时任务调度能在处理机之间随机地、方便地调度，达到各处理机负荷基本平衡，从而发挥系统的最大效能。因此，要求各个处理机具有相同功能，这就是同构型多处理机系统。例如，出现成百上千个微处理器构成的多处理机系统，其运算峰值速度可达到每秒上千万亿次。

要实现远距离多台计算机之间的资源共享，只有网络化，将通信功能从主机中分离出来，由专用通信处理机完成。计算机网络按其通信距离可划分为广域网（WAN）和局域网（LAN）。广域网距离远，通信速率较低。例如，由 ARPA 网发展而来的 Internet 就是典型的跨洲际的互连网络。局域网距离近，通信速率高，例如，智能化大楼内的计算机网络，如使用分布式

光纤数据连接的 FDDI (Fiber Dish Data Interface)、异步传输模式 ATM (Asynchronous Transfer Mode)、同步光纤网络 SONET (Synchronous Optical NETwork) 等技术可使通信速率超过 1000Mb/s。这已经接近多处理机的数据传输速率。局域网为分布处理系统的发展提供了许多有益的经验。分布处理系统的基础是局域网。

(3) 多机系统的比较。从图 5-1 可以看出, 无论单机系统还是多机系统, 按不同技术途径向三种不同类型的多处理机方向发展。前面已经介绍了各种类型多处理机的定义。它们之间的异同如表 5-1 所示。

表 5-1 多处理机系统比较

项 目	同构型多处理机	异构型多处理机	分布处理系统
目的	提高系统性能(可靠性、速度)	提高系统使用效率	兼顾效率与性能
技术途径	资源重复(机间互连)	时间重叠(功能专用化)	资源共享(网络化)
组成	同类型(同等功能)	不同类型(不同功能)	不限制
分工方式	任务分布	功能分布	硬件、软件、数据等各种资源的分布
工作方式	一个作业由多机协同并行地完成	一个作业由多机协同串行地完成	一个作业由一台处理机完成, 必要时才请求它机协作
控制形式	常采用浮动控制方式	采用专用控制方式	分布控制方式
耦合度	紧密耦合	紧密、松散耦合	松散、紧密耦合
对互连网络的要求	快速性、灵活性、可重构性	专用性	快速、灵活、简单、通用

由表 5-1 可见, 分布式系统与其他两类在概念上有交叉。就处理机之间的分工而言, 它既包含同构型多处理机的任务分布, 也包含异构型多处理机的功能分布。但三者工作方式是不同的。同构型多处理机是把一道程序(作业)分解成若干相互独立的程序段或任务, 分别由各个处理机并行执行。异构型多处理机是将作业分解成串行执行的若干任务, 分别由不同功能的处理机分工完成, 依靠流水作业的原理, 对多个作业重叠地进行处理。分布处理系统各处理机尽量完成本地作业, 当其资源和能力不够时才与其他处理机协同, 即较少地依赖集中的程序、数据和硬件。在系统控制方式上, 三者也不同。同构型多处理机常采用浮动控制方式, 即整个系统的管理由一台处理机控制, 但这台处理机不是固定地作为控制处理机, 其他处理机也可以承担, 这是因为这种类型的处理机具有相同的系统结构。异构型多处理机往往采用专用控制方式, 由一台专门处理机实现整个系统的集中控制。分布处理系统则采用分布式控制方式, 即由多台处理机协同完成系统的控制, 系统内部不存在明显的层次控制。其他差别, 可见表 5-1。

5.2 并行处理机基本结构

并行处理机也称 SIMD 计算机, 因为它是一个控制部件(CU)控制多个处理单元(PE)构成的阵列, 所以也称阵列处理机。SIMD 计算机主要用于大量高速向量或矩阵的运算场合。H.J.Siegel 提出了 SIMD 计算机操作模型, 如图 5-2 所示。

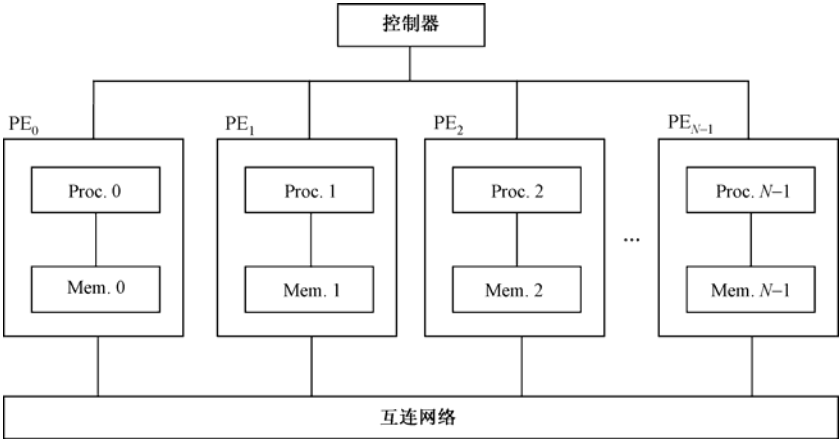


图 5-2 SIMD 计算机的操作模型

单指令流多数据流（SIMD）计算机的控制器管理多个处理单元 PE，所有 PE 均收到从控制器广播来的同一条指令，但操作对象是不同的数据。所以，SIMD 计算机的操作模型可用五元素组表示为

$$M = (N, C, I, M, R)$$

- (1) N 为机器的处理单元（PE）数。
 - (2) C 为由控制器（CU）直接执行的指令集，包括标量和程序流控制指令，即非广播执行的指令。
 - (3) I 为由 CU 广播至所有 PE 进行并行执行的指令集，例如算术运算、逻辑运算、数据查找、屏蔽以及其他由每个激活的 PE 对它的的数据所执行的局部操作。
 - (4) M 为屏蔽方案集，其中每个屏蔽方案将 PE 群划分为允许操作（激活）和禁止操作（休眠）两种状态。
 - (5) R 为数据寻径功能集，说明互连网络（ICN）中 PE 间通信所需要的各种设置模式。
- 例如，用五元素组描述 MasPar 公司的 MP-I 系列计算机特性如下：
- (1) MP-I 是 SIMD 机器， $N = 1024 \sim 16384$ ，PE 数与机器配置有关。
 - (2) CU 直接执行标量指令，对向量指令经译码后广播到 PE 阵列，并控制 PE 间通信。
 - (3) 每个 PE 都是基于寄存器的加载/存储型 RISC 处理机，PE 从 CU 接收指令，执行不同数据量的整数运算和标准浮点数运算。
 - (4) 屏蔽方案设在每个 PE 中，由 CU 连续监控，能在运行时动态调整每个 PE 状态，即置位（激活）或复位（休眠）。
 - (5) MP-I 由一个 X-Net 网络网络和一个全局多级交叉开关构成互连网络，可实现 CU-PE 间及一个 PE 与其 8 个近邻之间的通信，通过多级交叉开关实现全局通信。

SIMD 计算机的基本结构有两种：分布式存储器结构和共享式存储器结构。

5.2.1 分布式存储器结构

分布式存储器结构的 SIMD 计算机如图 5-3 所示。它有重复设置的多个同样的 PE，通过 ICN（Inter Connection Network）以一定方式相互连接。每个 PE（Processing Element）有各自的本地存储器 LM（Local Memory）。在一个阵列控制部件控制下，实现并行操作。程序和数据通过主机（前台机）装入控制存储器。指令由阵列控制部件译码；若是标量操作或控制

操作，则由标量处理机执行；若是向量操作，则广播到所有 PE 并行执行。数据集通过数据总线分布到所有 PE 的 LM，PE 所处理的数据从 LM 取得，并将结果写回到 LM。PE 通过 ICN 实现相互连接，控制部件通过执行程序来控制网络的寻径，完成点对点通信、广播、聚集等功能。PE 之间的同步是由控制部件硬件实现的，既保证各 PE 在同一周期内执行同一条指令，也可以通过对屏蔽位的设置，控制一个 PE 是否执行当前的指令。

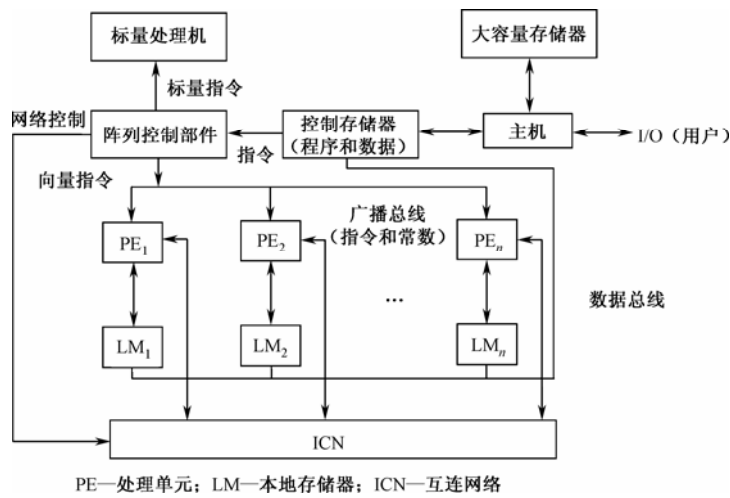


图 5-3 分布式存储器结构的 SIMD 计算机

从执行指令过程观察可知，SIMD 计算机每次只能执行一条指令，仍然是串行的，但是从执行向量数据过程观察，由于多个 PE 同时执行一条向量指令，产生多个数据流，因此具有数据并行性。目前的 SIMD 计算机几乎都是基于分布式存储器结构的系统，主要差别在于互连网络不同。

5.2.2 共享式存储器结构

共享式存储器结构的 SIMD 计算机如图 5-4 所示。它与分布式存储器结构的不同在于存储器结构上有明显区别。共享的多体并行存储器 SM（Shared Memory）通过 ICN 与各 PE 相连。SM 数目 \geq PE 数目，SM 模块为各 PE 共享，即任一个 PE 都可以访问任一个存储体。通过灵活、高速的 ICN，使 SM 与 PE 之间的数据传送以存储器的最高频率进行，使存储器冲突降到最低。ICN 由阵列控制部件根据控制指令译码进行控制。

这种共享式存储器结构在 PE 数目不太大的情况下是很理想的。BSP（Burroughs Scientific Processor）计算机就采用这种结构，它有 16 个 PE，17 个 SM，通过 16×17 ICN，可以实现无冲突并行地访问存储器。ICN 有完全交叉开关，以及用于广播（一个源 PE 广播至 n 个目的 PE）或共享（几个源 PE 寻找一个目的 PE）时化解冲突的硬件。在 PE 与 SM 数目互为质数时，就有可能实现无冲突访问。当 PE 增加时，为实现存储器分配互不冲突，将大量增加系统资源，从而大大降低系统性能价格比。

无论采用哪种存储器结构，ICN 都是必要的。在共享式内存方案中，它是 SM 与 PE 之间的必由之路。在分布式内存方案中，即使 PE 所需数据在大多数情况下由 LM 提供，而 PE 之间的数据通信仍是必不可少的。在图 5-3 中，各 PE 之间可以经过两条途径相互联系：一条

直接通过 ICN；另一条是数据从 LM 读至阵列控制部件，然后通过广播总线“广播”到所有 PE。在 PE 数目很多的并行处理机中，PE 之间的直接数据通路是很有限的，这就决定了并行处理机的系统固有结构和专用处理机的性质。这种局限性需通过对 ICN 的研究才能解决。

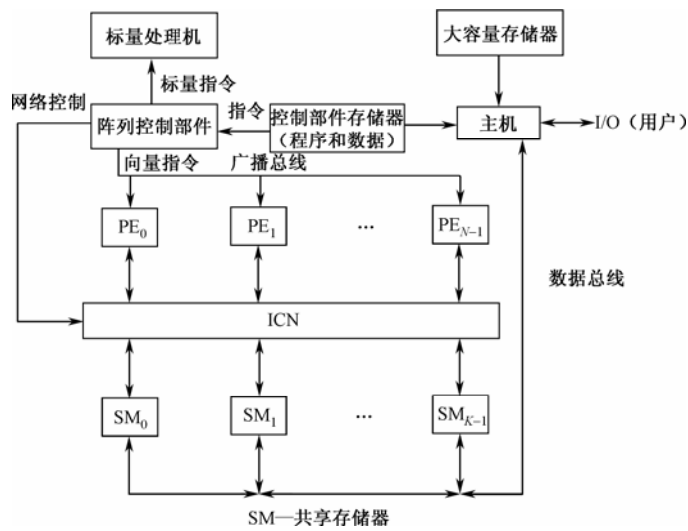


图 5-4 共享式存储器结构的 SIMD 计算机

5.2.3 并行处理机特点

SIMD 计算机主要特点如下：

（1）SIMD 计算机利用大量 PE 对向量的各分量同时进行计算，可获得很高的处理速度，所依靠的并行措施是资源重复，而不是时间重叠（如流水线处理机）。它属于并行性中的同时性，而不是并发性。在硬件价格大幅下降，系统结构不断改进之下，SIMD 计算机具有较好的性能价格比。

（2）SIMD 计算机最有特色的组成部分是它的互连网络（ICN），由于 ICN 规定了 PE 的连接模式，决定了 SIMD 计算机能适应的算法类别，对整个系统的各项性能指标产生了重要影响，在很好的 ICN 配合下，PE 阵列的功能和灵活性将会更强。

（3）SIMD 计算机适合于高速数值计算，类同于流水线向量处理机。它具有较固定的结构，直接与一定的算法相联系，其效率取决于计算程序向量化程度。应该把 SIMD 计算机的系统结构研究和算法研究结合起来，通过改进系统结构和研制并行算法，使其适应性更强，应用面更广。

（4）SIMD 计算机除向量运算速度以外，整个系统的实际有效速度还在相当程度上取决于标量运算速度和编译过程开销。所以，提高 SIMD 计算机处理标量和短向量（即向量元素少的向量指令）的能力是很重要的。同时，开发一个具有向量化功能的高级语言编译程序对提高 SIMD 计算机的通用性十分必要。

（5）SIMD 计算机基本上是一台向量处理的专用计算机，它必须和一台高性能的单处理器主机配合工作，由主机承担系统的全部管理功能。SIMD 的 PE 都是相同的，因此 PE 阵列是同构型的。但是，实际上的 SIMD 计算机是由 PE 阵列、标量处理机（构成后台机）和主机（前台机）按功能专用化原则组成的一个异构型计算机系统（见图 5-3 和图 5-4）。

5.3 并行处理机互连网络

互连网络（ICN）在 SIMD 计算机和 MIMD（Multiple Instruction Stream Multiple Data Stream，多指令流多数据流）计算机系统结构中占据重要地位，涉及互连网络作用、互连网络设计准则、互连函数、拓扑结构、性能参数、寻径机制等问题。

5.3.1 互连网络基本概念

互连网络是一种由开关元件按照一定的拓扑结构和控制方式构成的网络，用于计算机系统内部多个处理机或多个功能部件之间的相互连接。互连网络在多处理机系统中的位置和作用如图 5-5 所示。随着高性能计算的发展，SIMD 系统和 MIMD 系统的规模越来越大，处理机之间或处理单元和存储模块之间的通信要求和难度也越来越突出。所以，互连网络已成为并行处理系统的核心组成部分，对整个计算机系统的性能价格比有决定性影响。

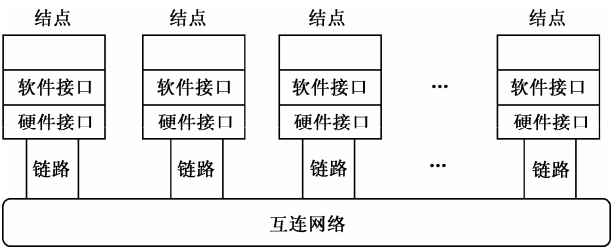
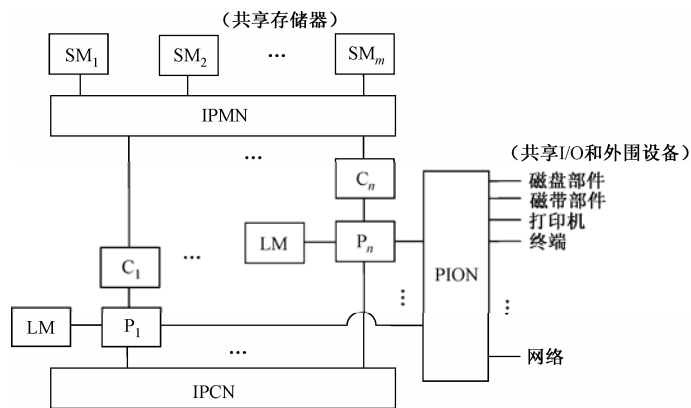


图 5-5 互连网络在系统中的位置和作用

图 5-6 是具有本地存储器 LM、本地高速缓冲存储器 C、共享存储器 SM 和共享外围设备的多处理机系统互连结构。每台处理机 P_i 与 LM_i 和 C_i 相连，构成一个结点，结点 i 与 SM_i 通过 IPMN 互连，结点 i 和共享外设通过 PION 互连，结点之间通过 IPCN 互连。

衡量互连网络性能主要取决于结点的度、网络直径、网络带宽、可靠性和成本等。在设计互连网络时应遵循以下准则：

- (1) 通信工作方式。它分为同步方式和异步方式。在同步方式中，PE 对数据进行并行操作或者 CU 对 PE 广播命令，都由统一时钟加以同步。SIMD 计算机多采用同步方式。异步方式不采用统一时钟，各处理机根据需要相互建立动态连接，MIMD 计算机多采用异步方式。
- (2) 控制策略。它分为集中和分散两种。集中式控制由一个控制器对所有互连开关状态加以控制，而分散式控制则由各个互连开关自行管理。SIMD 计算机多采用集中控制，而 MIMD 计算机两种方式都有。
- (3) 交换方式。它分为线路交换和分组交换两种。线路交换在通信过程中，在源和目的结点之间通过“呼叫-应答”建立通路，然后进行成批数据传送，在通信过程中源结点和目的结点独占通路，实时性好，通信信道利用率较低。分组交换则把要传送的数据按一定格式分成多个分组（数据包），分别送入互连网络，各分组可按不同路由到达目的结点，中间结点对收到的每个分组根据分组内指明的目的结点地址和当前网络状况予以路径选择，实行的是存储转发形式，所以通信过程开销大，但信道利用效率高。SIMD 计算机一般采用线路交换方式，因为 PE 间连接采用紧耦合。MIMD 多机系统则往往采用分组交换方式。



IPMN—处理机-存储器网络；PION—处理机-I/O网络；IPCN—处理机之间通信网络；
P—处理机；C—高速缓冲存储器；SM—共享存储器；LM—本地存储器

图 5-6 一般处理机系统的互连结构

（4）网络拓扑。它分为静态和动态两种。拓扑是指互连网络中各个结点间的连接关系，常用拓扑图描述。不同拓扑结构直接影响互连网络结构、通信形式、通信过程、通信控制方式等。

5.3.2 单级互连函数

可以把处理单元、存储器等看作结点，按照不同的拓扑关系，把 N 个入端和 N 个出端连接起来。为了反映互连网络的连接特性，可用一组互连函数来描述。当互连网络存在互连函数 f 时，则理力争

$$i = f(i) \quad (0 \leq i \leq N-1)$$

当互连网络用于实现处理器之间数据变换时，互连函数也反映了网络输入数组与输出数组间对应的置换关系或称排列关系，此时，互连函数也称为置换函数或排列函数。

表示互连函数一般有下列几种方法。

（1）互连网络中入、出端的连接图。一般在结点数较少的情况下使用。当结点数较多时，连接图非常复杂，难以体现各结点连接的内在规律。

（2）函数表示法。把所有入端 i 和出端 $f(i)$ 使用 n 位二进制编码表示，即 $X_{n-1}X_{n-2} \cdots X_1X_0$

$$X_{n-1}X_{n-2} \cdots X_1X_0 = f(X_{n-1}X_{n-2} \cdots X_1X_0)$$

（3）输入/输出对应表示法。入端和出端列表表示：

$$\begin{pmatrix} 0 & 1 & \cdots & N-1 \\ f(0) & f(1) & \cdots & f(N-1) \end{pmatrix}$$

基本互连函数及函数表达式如下。

1. 恒等互连网络

相同编号的输入端与输出端一一对应互连，其表达式为

$$I(X_{n-1}X_{n-2} \cdots X_1X_0) = X_{n-1}X_{n-2} \cdots X_1X_0$$

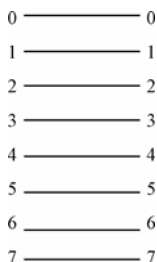
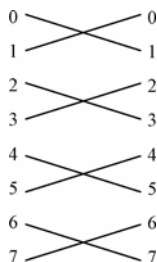
等式左边括号内为输入端二进制地址编码，等式右边为输出端二进制地址编码。其连接图如图 5-7 所示。

2. 交换互连网络

交换互连网络（Exchange）是实现二进制地址编码中第 0 位（ X_0 ）位值不同的输入端和输出端之间的连接。其表达式为

$$E(X_{n-1}X_{n-2}\cdots X_1X_0) = X_{n-1}X_{n-2}\cdots X_1\overline{X_0}$$

例如, 结点数 $N=8$, 则结点 0 (地址码 000) 与结点 1 (地址码 001) 相连, 结点 5 (地址码 101) 与结点 4 (地址码 100) 相连, 其余类似。其连接图如图 5-8 所示。


 图 5-7 $N=8$ 的恒等互连

 图 5-8 $N=8$ 的交换互连

3. 三维立方体单级互连网络

Cube 单级互连网络 (Cube) 如图 5-9 所示。立方体的每个顶点表示一个结点, 它有三种

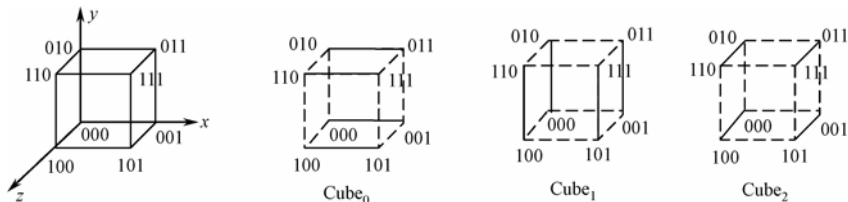
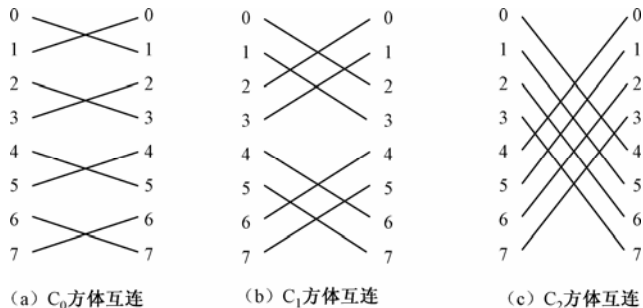


图 5-9 三维立方体互连网络

互连函数: Cube_0 , Cube_1 , Cube_2 。它的特点是 Cube 函数下标表示的入端和出端只在其位互为反码, 其他各位相同。例如 $\text{Cube}_0 (X_2X_1X_0) = X_2X_1\overline{X_0}$, $\text{Cube}_1 (X_2X_1X_0) = X_2\overline{X_1}X_0$, $\text{Cube}_2 (X_2X_1X_0) = \overline{X_2}X_1X_0$ 。推广至 n 维, Cube 单级互连网络共有 $n=\log_2 N$ 种互连函数 (N 为结点数, $N=2^n$)。其表达式为

$$\text{Cube}_i (P_{n-1}P_{n-2}\cdots P_i\cdots P_1P_0) = P_{n-1}P_{n-2}\cdots \overline{P_i}\cdots P_1P_0$$

($P_{n-1}P_{n-2}\cdots P_i\cdots P_1P_0$) 为结点二进制地址编码。式中, P_i 为入端标号的二进制地址码第 i 位, 且 $0 \leq i \leq n-1$ 。图 5-10 为 $N=8$ 的立方体互连。


 图 5-10 $N=8$ 的立方体互连

对于 n 维 Cube 互连网络, 要实现任意两个结点间的连接, 最多使用 n 次不同的互连函数, 因此, n 维 Cube 互连网络最大寻径距离为 n 。

4. 加减 2^i 单级互连网络 (PM2I, Plus-Minus 2^i)

PM2I 单级互连网络能实现 j 号结点与 $j \pm 2^i \bmod N$ 号结点的直接相连, 其互连函数为

$$\text{PM2}_{+i}(j) = j + 2^i \bmod N ; \quad \text{PM2}_{-i}(j) = j - 2^i \bmod N$$

式中, $0 \leq j \leq N-1, 0 \leq i \leq n-1, n = \log_2 N, N$ 为结点数。因此它只有 $2n$ 个互连函数。

例如, 对 $N=8$, 则各互连循环为

PM2₊₀: (0 1 2 3 4 5 6 7) ; PM2₋₀: (7 6 5 4 3 2 1 0)

PM2₊₁: (0 2 4 6) (1 3 5 7) ; PM2₋₁: (6 4 2 0) (7 5 3 1)

PM2_{±2}: (0 4) (1 5) (2 6) (3 7)

由于 PM2I 互连网络总存在 $\text{PM2}_{+(n-1)} = \text{PM2}_{-(n-1)}$, 所以实际上只有 $(2n-1)$ 个不同的互连函数。PM2I 互连网络最大寻径距离为 $\lceil n/2 \rceil$ 。在图 5-11 给出了 PM2₊₀, PM2₊₁, PM2₊₂ 连接图。而 PM2₋₀, PM2₋₁, PM2₋₂ 仅是图 5-11 内的箭头相反, 所以, PM2I 网络互连如图 5-12 所示 ($N=8$)。

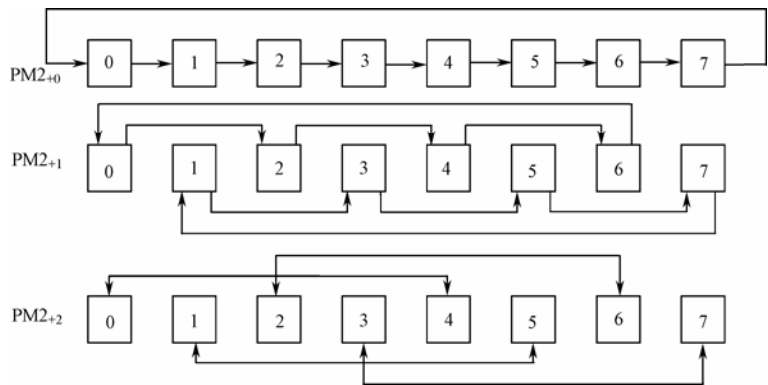


图 5-11 8 个结点的 PM2I 互连网络的部分连接图

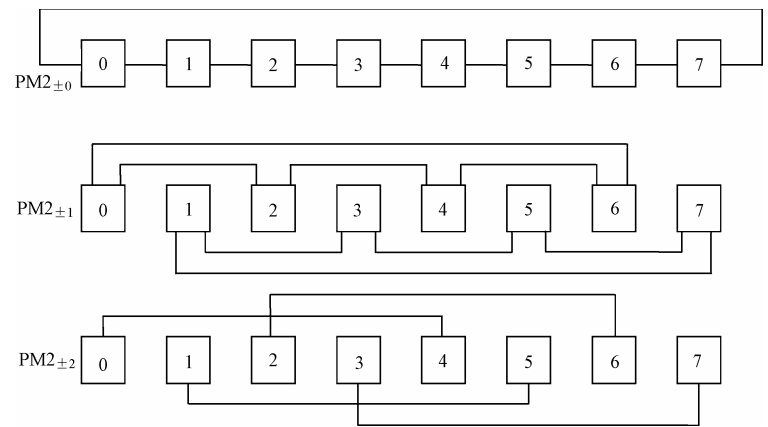


图 5-12 PM2I 网络

5. 混洗单级互连网络

混洗单级互连网络 (Shuffle) 是将输入端分成相等的两半, 前一半和后一半按序一个隔一个地从头至尾依次与输出端相连。有如洗扑克牌时, 将整幅牌均分两叠来洗, 达到一张隔一张的理想状态, 即均匀洗牌。其函数关系可表示为

$$\text{Shuffle}(P_{n-1}P_{n-2} \cdots P_1P_0) = P_{n-2} \cdots P_1P_0P_{n-1}$$

在此互连网络中, 经过 $n=\log_2 N$ 次混洗连接后, 除了地址码为全 0 和全 1 的结点外, 其余结点都有与其他结点连接的机会。图 5-13 为 $N=8$ 个结点的全混洗连接图。

由于 Shuffle 互连网络不能实现地址码为全 0 和全 1 的结点与其他结点连接, 因此, 在这个网络内增加交换互连函数 Cube_0 构成混洗交换 (Shuffle-Exchange) 单级互连网络, 如图 5-14 所示。图中虚线表示 Shuffle, 实线表示 Exchange。最远的两个结点间连接需要经过 n 次交换和 $(n-1)$ 次混洗, 所以最大寻径距离为 $(2n-1)$ 。

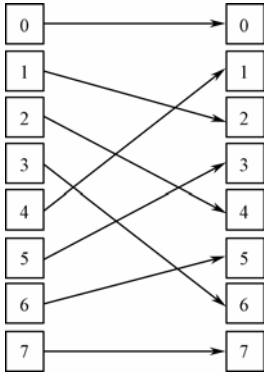


图 5-13 8 个结点的全混洗连接图

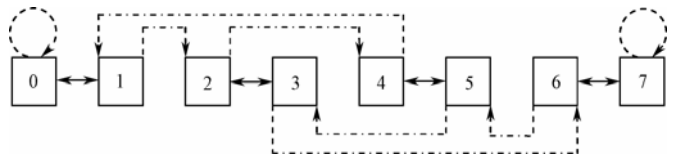


图 5-14 8 个结点的全混洗交换网络连接图

将结点分成若干子组, 每个子组完成混洗, 就形成了子混洗, 其互连函数为

$$\text{SubShuffle}(P_{n-1}P_{n-2}\cdots P_{i+1}P_i P_{i-1}\cdots P_1P_0) = P_{n-1}P_{n-2}\cdots P_{i+1}P_{i-1}\cdots P_1P_0 P_i$$

$N=8$ 时, 子混洗连接如图 5-15 (b) 所示。

如果以若干结点为一组, 然后进行混洗, 得到超混洗, 其互连函数为

$$\text{SuperShuffle}(P_{n-1}P_{n-2}\cdots P_{n-i}P_{n-i-1} P_{n-i-2}\cdots P_1P_0) = P_{n-2}\cdots P_{n-i}P_{n-i-1} P_{n-1} P_{n-i-2}\cdots P_1P_0$$

$N=8$ 时, 超混洗连接如图 5-15 (c) 所示。

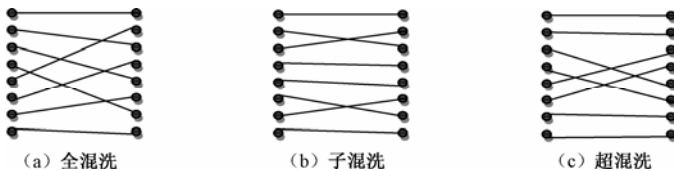


图 5-15 混洗网络

逆混洗是 Shuffle 的逆函数, 其函数表达式为

$$\text{逆 Shuffle}(P_{n-1}P_{n-2}\cdots P_1P_0) = P_0 P_{n-1}P_{n-2}\cdots P_1$$

$N=8$ 时, 逆混洗连接如图 5-16 所示。混洗和逆混洗是两种十分有用的互连函数, 以它们为代表结合交换开关多级组合可构成 Ω 网络或逆 Ω 网络。它已在多项式求值、矩阵转置和 FFT 等并行运算及并行排序等方面得到广泛应用。

6. 蝶形单级互连网络

蝶形单级互连网络 (Butterfly) 源于 FFT 变换, 将输入端二进制地址码的最高位和最低位互换位置即可求得相应的输出端地址码, 其互连函数为

$$\text{Butterfly}(P_{n-1}P_{n-2}\cdots P_1P_0) = P_0P_{n-2}\cdots P_1 P_{n-1}$$

图 5-17 为 $N=8$ 的蝶形互连网络连接图。

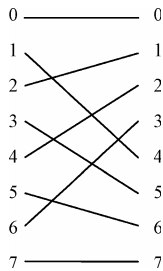


图 5-16 $N=8$ 的逆混洗连接

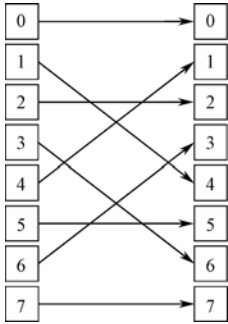


图 5-17 8 个结点的蝶形互连网络连接图

同样可定义子蝶式，其互连函数为

$$\text{SubButterfly}(P_{n-1}P_{n-2}\cdots P_{i+1}P_i P_{i-1}\cdots P_1P_0) = P_{n-1}P_{n-2}\cdots P_{i+1} P_0P_{i-1}\cdots P_1 P_i$$

超蝶式（SupperButterfly）互连函数如下：

$$\text{SupperButterfly}(P_{n-1}P_{n-2}\cdots P_{n-i}P_{n-i-1} P_{n-i-2}\cdots P_1P_0) = P_{n-i-1}P_{n-2}\cdots P_{n-i} P_{n-1} P_{n-i-2}\cdots P_1P_0$$

7. 位序颠倒单级互连网络

位序颠倒单级互连网络（Reverse-Arrangement）是将输入端二进制地址码位序倒过来排序作为输出端二进制地址码。其互连函数如下：

$$\text{Reverse}(P_{n-1}P_{n-2}\cdots P_1P_0) = P_0 P_1\cdots P_{n-2} P_{n-1}$$

同样可以定义子位序颠倒，其互连函数为

$$\text{SubReverse}(P_{n-1}P_{n-2}\cdots P_{i+1}P_i P_{i-1}\cdots P_1P_0) = P_{n-1}P_{n-2}\cdots P_{i+1} P_0 P_1\cdots P_{i-1}P_i$$

超位序颠倒互连函数为

$$\text{SupperReverse}(P_{n-1}P_{n-2}\cdots P_{n-i}P_{n-i-1} P_{n-i-2}\cdots P_1P_0) = P_{n-i-1} P_{n-i} \cdots P_{n-2}P_{n-1}P_{n-i-2}\cdots P_1P_0$$

对于 $N=8$ 的情况，蝶式和位序颠倒的连接图是一样的，如图 5-18 所示。

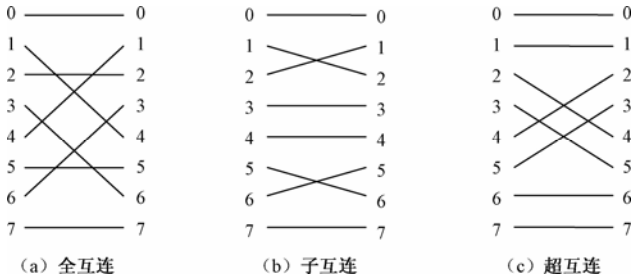


图 5-18 $N=8$ 的蝶式互连和位序颠倒互连

8. 移位单级互连网络

移位单级互连网络（Move）是将输入端 X 号结点与 $(X+K) \bmod N$ 号结点直接相连，其互连函数为

$$M(X) = (X+K) \bmod N \quad (0 \leq X \leq K)$$

式中， K 为常数，指移动的位置值。也可以将整个输入结点分成若干子结点，在子结点内进行循环移位互连，其互连函数为

$$M(X)_{(n-1),i} = (X)_{(n-1),i}$$

$$M(X)_{(i-1),0} = ((X)_{(i-1),0} + K) \bmod 2^i$$

式中, 下标 $(n-1), i$ 和 $(i-1), 0$ 分别指从 $(n-1)$ 到 i 位和从 $(i-1)$ 到 0 位。移位互连如图 5-19 所示。

PM2I 互连实为移位互连中的一种, 其 $K=+2^i$ 或 $K=-2^i$ 。图 5-20 所示为 PM2_{+i} 的互连图。

5.3.3 互连网络特性

网络可用有向边或无向边连接有限个结点的拓扑图来表示。网络特性如下:

(1) 网络规模。网络中结点的总数, 反映网络连接的部件多少。

(2) 结点度 (Node Degree)。与结点相连接的边 (即链路或通道) 数称为结点度。在单向通道情况下, 进入结点的通道数称为入度 (In Degree), 而从结点出来的通道数则称为出度 (Out Degree)。结点度为入度和出度之和, 它反映了结点所需要的 I/O 端口数, 也反映了结点的价格。若为降低网络价格, 应尽可能使结点度小。若为使网络可扩展, 构件能模块化, 则要求结点度保持恒定。

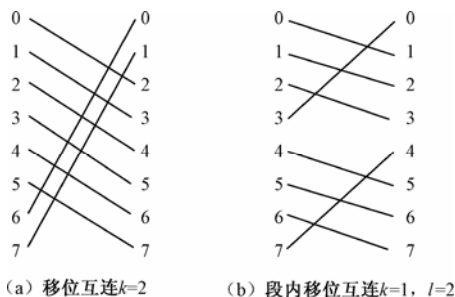


图 5-19 $N=8$ 的移位互连

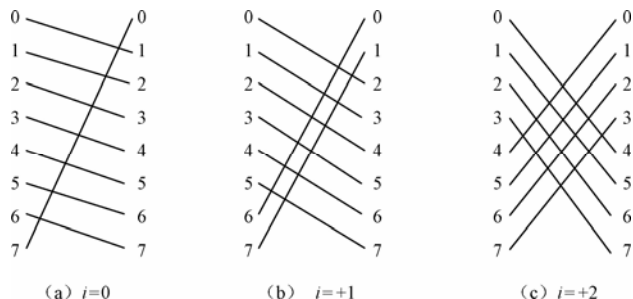


图 5-20 $N=8$ 的 PM2I 互连

(3) 距离。两个结点间相连的最少边数。

(4) 网络直径 (Network Diameter)。它是网络中任意两个结点之间距离的最大值。这是网络通信性能的一个指标。从通信的角度来看, 网络直径应当尽可能地小。

(5) 聚集带宽 (Aggregate Bandwidth)。从一半结点到另一半结点, 每秒传输的最大位数 (b/s) 或字节数 (B/s)。例如 HPS 是一个对称式网络, 共有 512 个结点, 每个结点端口带宽为 40MB/s, 则 HPS 的聚集带宽为 $512/2 \times 40\text{MB/s} = 10\text{GB/s}$ 。

(6) 等分带宽 (Bisection Bandwidth)。当某一网络被分成相等的两半时, 沿分界面的最小边数 (通道) 称为通道等分宽度。一个网络可以有多个等分平面, 最小等分平面是指具有最小连线数的等分平面。网络的等分带宽指每秒在最小等分平面上通过所有连线的最大位数或字节数。设 b 为最小等分平面的链路数, w 为每条链路的连线数, r 为每条连线的最大的数据传输速率 (单位 b/s), 则该网络的等分带宽 $B=bwr(\text{b/s})$ 。

(7) 结点间的线长。它是两个结点间连线的长度。它会影响信号在信道上传播时延、时钟信号变形和对功率的需要。

(8) 网络对称性 (Network Symmetry)。如果从网络任何结点看出来的拓扑结构都是一样

的，则称该网络为对称网络，否则称为非对称网络。对称网络较易实现，可扩展性好，寻径效率较高，编程也较容易。

网络通信是将源结点的消息（报文）传输到目的结点的过程。源结点作为发送方，其操作步骤如下：

- (1) 发送程序把报文传送到发送缓冲区。
- (2) 计算待发送报文的校验位，并附在报文之后，按通信协议构造成规定的格式（帧、分组等格式）。
- (3) 将帧或分组通过网络接口硬件在信道上发送。
- (4) 发送结束后启动定时计数器，等待接收方确认。
- (5) 在定时范围内收到接收方确认，则发送方结束本次通信。若超时未收到接收方确认，则发送方重发报文。

目的结点作为接收方，其操作步骤如下：

- (1) 接收程序把报文从网络接口硬件取到接收缓冲区。
- (2) 对收到的报文进行校验。若正确，则向发送方发送确认，并把报文传送到用户程序缓冲区，由用户程序处理。
- (3) 若出错，接收方将收到的报文丢弃，等待发送方超时重发。

涉及网络通信方面有关时延的性能参数如下：

- (1) 频宽（Bandwidth）：指网络传输信息的最大速率，也称传输速率，单位是 b/s。
- (2) 传输时间（Transmission Time）：消息（报文）通过网络的时间，它等于消息（报文）长度除以频宽。

(3) 传播时延（Spread Latency）：报文的第一位到达接收方所用的时间，它包括网络中转结点转发及其他硬件所带来的时延。一般认为，信号在导体中传输的速度大约为 200m/μs（即 5μs/km）。

(4) 传输时延（Transport Latency）：它等于传输时间和传播时延之和，它是报文在网络上传输所用去的时间。

(5) 发送方开销（Sender Overhead）：发送方通信处理器把消息（报文）送上网络的时间，即发送方软、硬件所用时间。

(6) 接收方开销（Receiver Overhead）：接收方通信处理器从网络上把到达的消息（报文）取出的时间，即接收方软、硬件所用时间。所以，一个消息（报文）在网络上通信的总时间为

$$\text{总时间} = \text{发送方开销} + \text{传播时延} + \frac{\text{消息长度}}{\text{频宽}} + \text{接收方开销}$$

例如，某网络传输速率为 10Mb/s，发送方开销为 230μs，接收方开销为 270μs，若两个结点相距 500m，现发送 1 个 1000B 的消息，则总时间为多少？若两个结点相距 100km，则总时间为多少？

解：相距 500m 时，总时间

$$\begin{aligned} T &= 230 + \frac{500}{200} + \frac{1000 \times 8}{10 \times 10^6} + 270 \\ &= 230 + 2.5 + 800 + 270 \\ &= 1302.5\mu\text{s} \end{aligned}$$

相距 100km 时, 总时间

$$T = 230 + \frac{100 \times 10^3}{200} + \frac{1000 \times 8}{10 \times 10^6} + 270 = 1800 \mu\text{s}$$

5.3.4 静态互连网络

静态互连网络是指各结点间有专用的链路, 且在运用中不能改变的网络。在此类网络中, 每一个开关元件固定与一个结点相连, 建立该结点与邻近结点间的连接通路, 直接实现两个结点之间的通信。静态互连网络适合于构造通信模式可预测的计算机, 或者用静态连接实现通信的计算机。静态互连网络拓扑结构有一维、二维、三维及三维以上, 各种拓扑结构的静态网络如下所述:

(1) 线性阵列。这是一种一维网络。 N 个结点用 $N-1$ 条链路连成一行, 如图 5-21 (a) 所示。内部结点度为 2, 端结点度为 1, 网络直径为 $N-1$ 。等分宽度 $b=1$ 。线性阵列有最简单的拓扑结构, 这种结构不对称, 当 N 很大且结点相距较远时, 通信效率很低。线性阵列与总线的区别在于: 线性阵列允许不同的源结点和目的结点对并发使用系统的不同的通道, 而总线是通过切换结点来实现时分特性的。

(2) 环和带弦环 (Chordal Ring)。将线性阵列的两个端结点连接起来就得到环拓扑, 如图 5-21 (b) 所示。环可以单向工作或双向工作。它是对称的, 结点度是常数 2。单向环直径为 $N-1$, 双向环直径为 $N/2$ 。若将结点度提高至 3 或 4, 就可得到图 5-21 (c) 和 (d) 所示的带弦环。双向环网络直径为 8 ($N=16$), 度 3 的带弦环网络直径为 5, 度 4 的带弦环网络直径为 3。所以, 增加环上结点的链路 (即弦), 则结点度增大, 网络直径减小。在极端情况下, 如图 5-21 (f) 所示全连接网 (Completely Connected Network), 结点度为 15 ($N=16$) (即 $N-1$), 网络直径最短, 为 1, 链路数为 120 (即 $N(N-1)/2$)。

(3) 循环移数网络 (Barrel Shifter)。图 5-21 (e) 所示 $N=16$, 将环上每个结点到与其距离为 2 的整数幂的结点之间增加一条附加链就构成了循环移数网络。如果 $j-i=2^r$, $r=0, 1, 2, \dots, n-1$, 网络结点数为 $N=2^n$, 则结点 i 与结点 j 连接, 结点度为 $2n-1$, 网络直径为 $n/2$ 。如图 5-21 (e) 所示循环移数网络 $N=16=2^4$, $n=4$, 则结点度为 7, 网络直径为 2。该种网络复杂性比全连接网络低。

(4) 树形和星形。5 层 31 个结点的二叉树如图 5-22 (a) 所示。 n 层全平衡的二叉树应有 $N=2^n-1$ 个结点, 最大结点度为 3, 直径是 $2(n-1)$ 。由于结点度是常数, 因此二叉树是一种可扩展的系统结构, 但其网络直径相当长。

星形是一种 2 层树, 结点度高, 为 $N-1$ 。网络直径小, 为 2。图 5-22 (b) 是星形拓扑图。星形结构常用于集中监控系统。

(5) 胖树形 (Fat Tree)。二叉胖树形结构如图 5-22 (c) 所示。由 Leiserson 于 1985 年将一般树结构修改成胖树形。胖树的通道从叶结点向根结点上行, 其宽度逐渐增宽, 由此缓解传统二叉树通向根结点的瓶颈问题。

(6) 网格形和环网形。图 5-23 (a) 所示为 3×3 网格形网络。当网格形网络结点数为 $N=n^k$ 时, 称为 k 维网格, 内部结点度为 $2k$, 网络直径为 $k(n-1)$ 。图 5-23 (a) 所示纯网格形是不对称的, 边结点的结点度为 3, 角结点的结点度为 2, 内结点的结点度为 4。

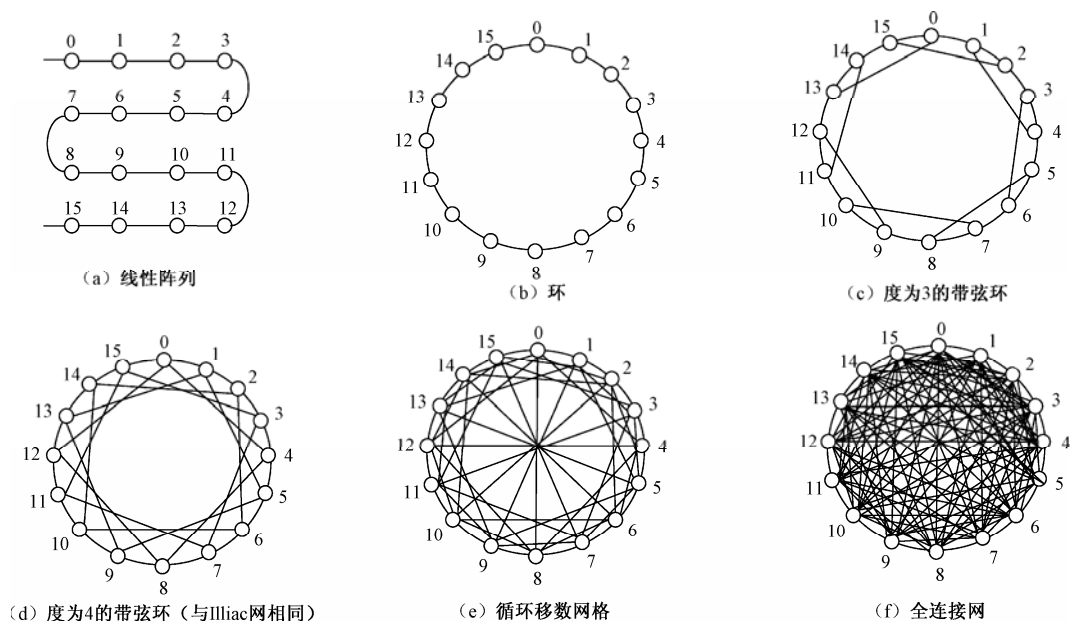


图 5-21 线性阵列、环、带弦环、循环移数网格、全连接网的拓扑结构

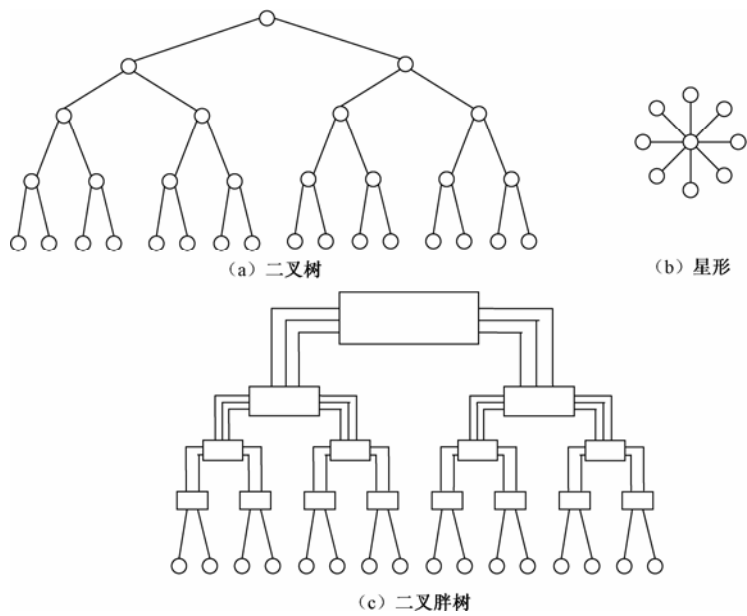


图 5-22 树形、星形和胖树形网络

图 5-23 (b) 是一种回绕连接的网格图，是 Illiac 网使用的结构。Illiac IV 系统采用 8×8 回绕网，其结点度为 4，网络直径为 7。若 $N=16=4 \times 4$ Illiac 网，与图 5-21 (d) 带弦环在拓扑上是等效的。 $n \times n$ 的 Illiac 网的直径为 $n-1$ ，仅为纯网格直径的一半。环形网络是将环形和网络结合在一起，其网络直径更短，并能向高维扩展，如图 5-23 (c) 所示。一个 $n \times n$ 二元环网形网络结点度为 4，网络直径为 $2 \times (n/2)$ 。环网形是一种对称拓扑结构，回绕连接使其直径比网格结构减少二分之一。

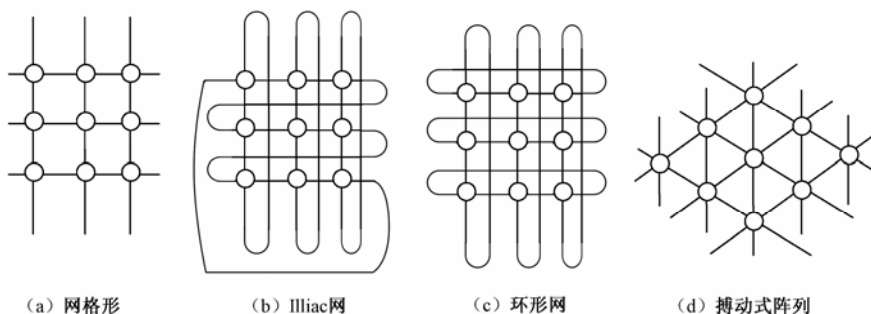


图 5-23 网格形、Illiac 网、环网形和搏动式阵列

(7) 搏动式阵列。这是一类为实现确定算法而设计的多维流水线阵列结构。图 5-23 (d) 所示就是为实现二个矩阵相乘而专门设计的搏动式阵列。该阵列内部结点度为 6。静态搏动式阵列可以在多个方向上使数据流变成以流水线方式工作。通过确定的互连和同步操作，搏动式阵列可与算法的通信结构相匹配。在信号处理、图像处理等应用领域，搏动式阵列的性能价格比更好。但是，其结构专用性强，实用性有限，编程很难。自从 1978 年 Kung 和 Leiserson 提出搏动式阵列后，它已成为一个广泛研究的领域。

(8) 超立方体。它是二元 n 维立方体。 n 维立方体由 $N=2^n$ 个结点组成，分布在 n 维上。 $N=8=2^3$ 立方体如图 5-24 (a) 所示，每维有二个结点，称为二元。4 维立方体可以通过将两个三维立方体相应结点互连组成，如图 5-24 (b) 所示。 n 维立方体的结点度等于 n ，也就是网络直径。结点度随维数线性地增加，所以超立方体很难扩展，难以组成高维立方体。

(9) 带环立方体。它是从超立方体改进而来的。将三维立方体的角结点（顶角）用三个结点的环代替，就成为带环三维立方体 (CCC)，如图 5-24 (c) 所示。将 K 维立方体的每个结点用 K 个结点组成的环替代，即 K 维超立方体的每个顶角用有 K 个结点的环取代，就构成了带环 K 维立方体，如图 5-24 (d) 所示。该图左边表示顶角有 K 维（即 K 个方向），该图右边表示顶角上有 K 个结点组成的环。带环 K 维立方体共有 2^K 个结点环。这样一个 K 维立方体可演变成 $K \times 2^K$ 个结点的 K -CCC 网络。图 5-24 (c) 所示 3-CCC 的网络直径为 6，比三维立方体网络直径大一倍。 K -CCC 网络直径为 $2K$ 。CCC 的结点度为常数 3，与超立方体维数无关。设某超立方体有 $N=2^n$ 个结点。一个有相同 N 结点数的 CCC 一定由低维 K 立方体组成，即 $2^n = K \times 2^K$ ， $K < n$ 。例如， $N=64$ ，可用六维超立方体组成（即 $N=64=2^6$ ， $n=6$ ），也可以用 4-CCC 组成，即 $K=4$ 维，每个顶角用 4 个结点构成的环。4-CCC 的网络直径为 $2K=2 \times 4=8$ ，而六维超立方体网络直径为 6，但 CCC 的结点度为 3，比六维超立方体结点度 6 要小。所以，若允许一定时延，则 CCC 是一种构造可扩展系统的较好的结构。

(10) K 元 n 维立方体网络。环形、网格形、环网形、二元 n 维立方体（超立方体）和 Ω 网络都是 K 元 n 维立方体网络系列的拓扑同构体。图 5-25 所示就是四元三维立方体网络。 n 是立方体的维数， K 是基数或者说是每个方向的结点数（即多重性）。 n ， K 与网络中总结点数 N 的关系为

$$N=K^n \quad \left(K = \sqrt[n]{N}, n = \log_K N \right)$$

K 元 n 维立方体的结点地址用基数为 K 的 n 位表示，即 $A=a_1a_2 \cdots a_n$ ，其中 a_i 表示第 i 维结点位置。 K 元 n 维立方体网络内所有链路都认为是双向的。低维 K 元 n 维立方体称为环网，高维二元 n 维立方体称为超立方体。

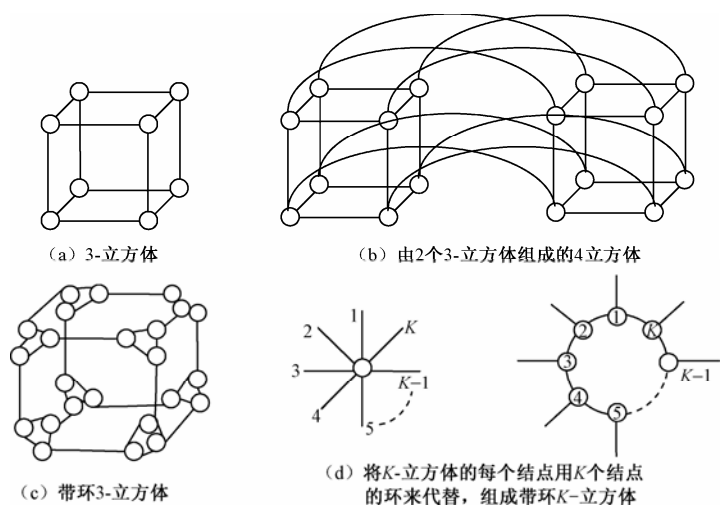


图 5-24 超立方体和带环立方体

用网络折叠的方法可避免环网中长的端绕连接，如图 5-26 所示。当多维网络装入一个平面时，每维上沿环的所有链路的线长相等。这种网络价格取决于连线量而不是开关数。在线等分为常数的前提下，宽通道低维网络比窄通道高维网络延迟较低、冲突较少、热点吞吐量较大。

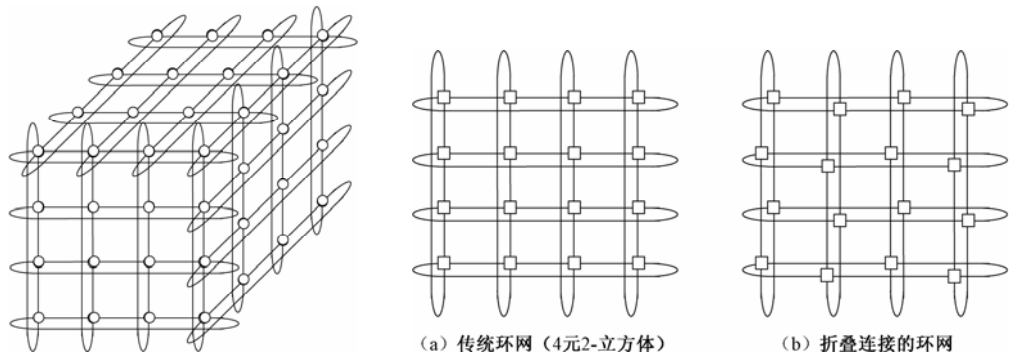


图 5-25 $K=4$ 和 $n=3$ 的 K 元 n 维立方体网络

图 5-26 环网中线长相等的折叠连接

表 5-2 是静态互连网特性一览表。表中多数网络结点的度小于等于 4，这是比较理想的。全连接网络和星形网络结点度太高。超立方体的结点度随 $\log_2 N$ 值增大而增大，当 N 值很大时，其结点度也太高。网络直径变化范围很大，但随着硬件寻径技术提高，在高度流水线操作下，任意两个结点间的通信延迟几乎固定不变。链路数会影响网络价格，等分宽度将影响网络带宽，对称性会影响可扩展性和寻径效率，网络总价格随结点度和链路数增大而上升。网络直径小是评价指标之一，结点间平均距离是更确切的量度指标。等分宽度可用提高通道宽度来扩大。根据上述分析，环、网格、环网、 K 元 n 维立方体和 CCC 都具有构造大规模并行处理机 MPP (Massively Parallel Processor) 的条件。

表 5-2 静态网络特性一览表

网络类型	结点度 d	网络直径 D	链路数 l	等分带宽 B	对称性	网络规格评注
线性阵列	2	$N-1$	$N-1$	1	非	N 个结点
环形	2	$\lceil N/2 \rceil$	N	2	是	N 个结点
全连接	$N-1$	1	$N(N-1)/2$	$(N/2)^2$	是	N 个结点
二叉树	3	$2(h-1)$	$N-1$	1	非	树高 $h=\lceil \log_2 N \rceil$
星形	$N-1$	2	$N-1$	$\lceil N/2 \rceil$	非	N 个结点
二维网络	4	$2(r-1)$	$2N-2r$	r	非	$r \times r$ 网络, $r=\sqrt{N}$
Illiac 网	4	$r-1$	$2N$	$2r$	非	与 $r=\sqrt{N}$ 的带弦环等效
二维环网	4	$2\lceil r/2 \rceil$	$2N$	$2r$	是	$r \times r$ 网络, $r=\sqrt{N}$
超立方体	n	n	$nN/2$	$N/2$	是	N 个结点, $n=\log_2 N$ (维数)
CCC	3	$2K-1+\lceil K/2 \rceil$	$3N/2$	$N/(2K)$	是	$N=K \times 2^k$ 结点, 环长 $K \geq 3$
K 元 n -立方体	$2n$	$n\lceil K/2 \rceil$	nN	$2K^{n-1}$	是	$N=K^n$ 个结点

注：表中 $\lceil \quad \rceil$ 表示取整运算。

5.3.5 动态互连网络

动态互连网络可以达到多种用途和通用目的，它可以根据程序要求实现所有通信模式，它使用开关或仲裁器以提供动态连接特性。动态互连网络的价格和所采用的链路、开关、仲裁器的成本有关，其性能涉及网络带宽、数据传输速率、网络时延和所用的通信模式。常见的动态互连网络有总线互连、交叉开关互连、多级互连网络、蝶形网络和组合网络等。

1. 总线互连

总线互连方式是多机系统中实现互连的最简便的一种结构形式。多个处理器、存储器模块和 I/O 部件通过各自的接口与系统总线相连，也可以由 CPU、局部存储器、I/O 部件构成计算机模块，多个计算机模块通过公共接口部件与系统总线相连。总线通信方式采用分时或多路转换方式，实现信息在主设备和从设备之间的传输。能实现多机互连的总线标准有 PCI，VME，MULTIBUS，SBUS，MICROCHANNEL，IEEE Futurebus 等。标准总线在构建单机系统时，价格低廉。而多处理机总线和层次型总线常用来构建 SMP（Symmetric Multiprocessor，对称多处理机）、DSM（Distributed Shared Memory Multiprocessor，分布共享存储器多处理机）、NUMA（Non Uniform Memory Access，不一致存储器存取）机器。这些可扩展的总线一般用硬件支持 Cache 一致性、多处理机同步、中断处理等。

典型多处理机总线结构如图 5-27 所示。系统总线在底板或中央板上。CPU 或 IOP（I/O 处理器）是主设备，存储器或盘驱动器是从设备。各模块（卡或板）通过接口逻辑挂上系统总线。局部存储器 LM、I/O 控制器 IOC、存储控制器 MC、通信控制器 CC 使用在不同插接板上，发挥各自作用。在各个板的内部都有各自的卡（板）内总线。

多处理机总线要解决的主要问题有总线仲裁、中断处理、协议转换、快速同步、Cache 一致性、分离事务、总线桥接和层次总线扩展等。硬连接路障同步线在机群总线中起到同步提速作用，将原来基于软件的同步时间从几千微秒降低到几百纳秒。

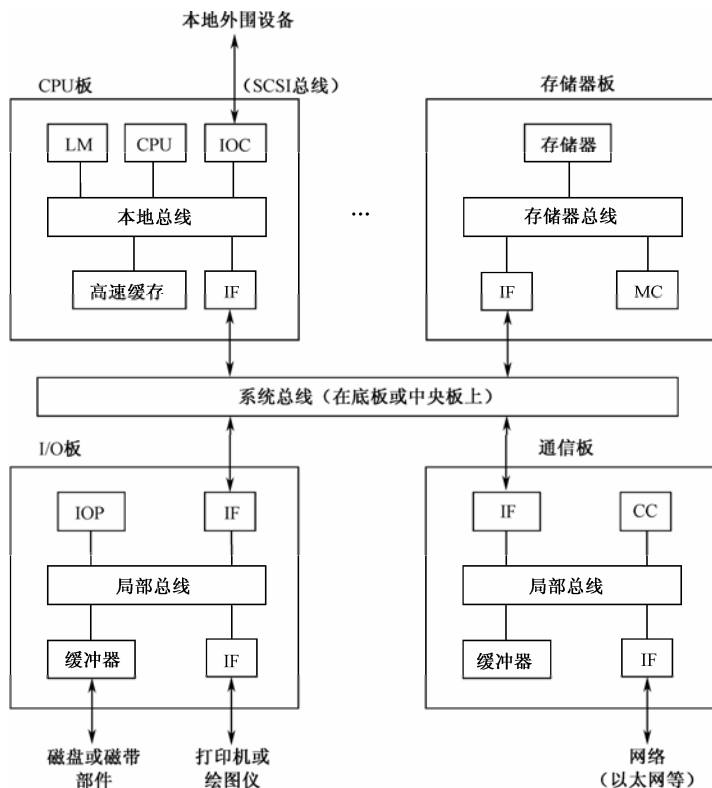


图 5-27 多处理机总线结构

SMP 总线在构建大规模系统时可扩展性不足，使用层次总线结构有助于增加可扩展性。图 5-28 所示 CC-NUMA (Cache Coherent – Non Uniform Memory Access, Cache 群不一致存储器存取) 机器设计方法，使用层次总线互连多个多处理机机群构成系统。属于同一机群的处理器 P 各自有本地 Cache，为一级 Cache，通过机群总线互连。机群高速缓存 (CC) 作为二级 Cache，供机群中各个 P 共享。各个机群通过连接全局的机群间总线互连，相互通信，并共享存储器模块。层次总线必须用网桥作为机群间接口，以保持本地和共享 Cache 的一致性。IEEE Futurebus 总线为构造层次总线提供专用网桥、Cache、MEM 代理、消息接口和电缆分段等机制。

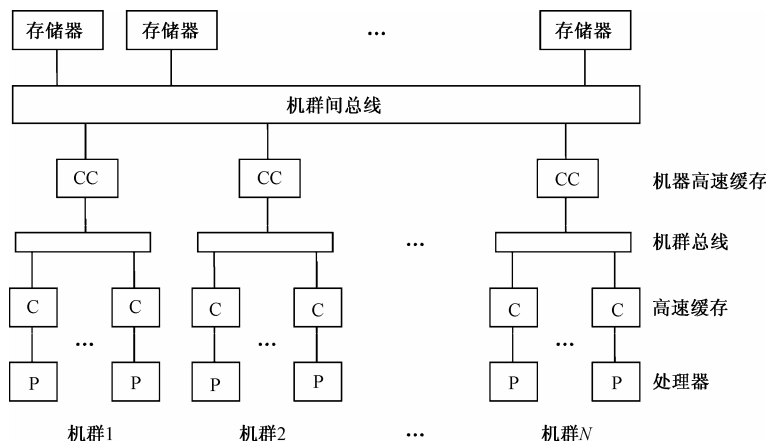


图 5-28 连接多处理机机群构造 CC-NUMA 机器的层次总线

利用总线互连虽然简便，但也存在下述缺点：① 总线实行分时共享，即使总线带宽很高，被多个处理器平均一分，每个处理器带宽也就有限。② 总线缺少冗余机制，易于出错。③ 总线一般局限于较小的机架内，使用层次总线扩展到几个机架时，时钟扭曲和全局定时就成了问题，所以总线可扩展性有限。

2. 交叉开关互连

交叉开关（Crossbar）是一个单级交换网络，只允许一对一置换（映射），即开关只有两种连接模式：直送和交换。交叉开关可以实现所有结点之间无阻塞连接，可为每个端口提供更高的带宽。与总线互连中采用分时使用总线不同，交叉开关采用的是空间分配机制。在并行处理中，交叉开关有两种使用方式：一种用于对称式多处理机或多计算机群中的处理器间通信；另一种用于 SMP 服务器或向量超级计算机中处理器和存储器之间的存取操作。

一个 $a \times b$ 开关模块有 a 个输入和 b 个输出，一个二元开关的 $a=b=2$ 。理论上 a 和 b 不一定相等，而实际上 a 和 b 经常选为 2 的整数幂，即 $a=b=2^K$ ， $K \geq 1$ 。在 $a \times b$ 开关中，每个输入可与一个或多个输出相连，但是不允许多个输入与一个输出相连，即一对一和一对多映射是允许的，多对一映射是不允许的，因为输出端将发生冲突。一个二元开关有 4 种合法状态：直送、交换、上播、下播，如图 5-29 所示。这样的开关称为交换开关，而交叉开关只有直送和交换。一个 $n \times n$ 交叉开关，其合法状态有 n^n 个，置换连接有 $n!$ 个，见表 5-3 所示。

交叉开关互连由一组二维阵列的开关组成，如图 5-30 所示。它将横向的处理器 P 与纵向的存储器模块 M 连接起来。阵列中的总线条数等于 $p+m$ ，只要 $m \geq p$ ，就可以使每个处理器都能通过总线与某个存储器模块相连，从而加宽了带宽。图 5-30 中每个交叉点都是一套开关，开关内有多路转换逻辑和仲裁部件。若 p 个处理器还要和 i 个 I/O 模块相连，这时阵列中总线条数为 $p+m+i$ ，只要 $m \geq p+i$ ，就能实现无阻塞连接。

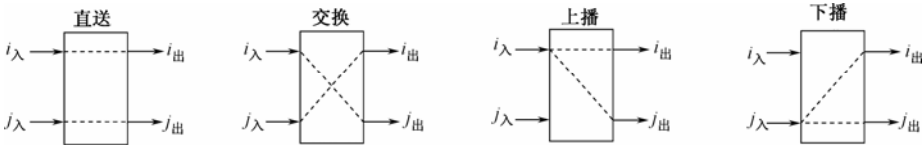


图 5-29 交换开关的 4 种连接方式

表 5-3 开关模块和合法状态

模块大小	合法状态	置换连接
2×2	4	2
4×4	256	24
8×8	16777216	40320
$n \times n$	n^n	$n!$

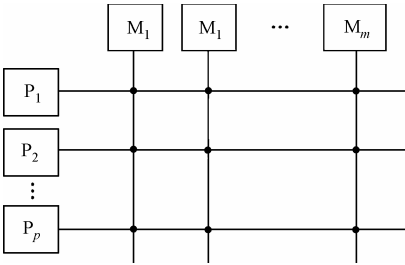


图 5-30 交叉开关互连

5.3.6 多级互连网络

多级互连网络（Multistage Interconnection Network, MIN）是指将多套单级互连网络通过交换开关或交叉开关串联扩展而成的网络。由于可以改变开关的控制方式（即开关连接方式）

灵活地实现各种连接，满足系统应用的需要，因此在 SIMD 和 MIMD 计算机设计中较多使用了 MIN。MIN 在每级上使用多个 $a \times b$ 开关模块，相邻级间开关之间使用固定的级间连接，其结构模型如图 5-31 所示。为了在输入和输出之间建立所需的连接，可以用动态设置开关的状态来实现。

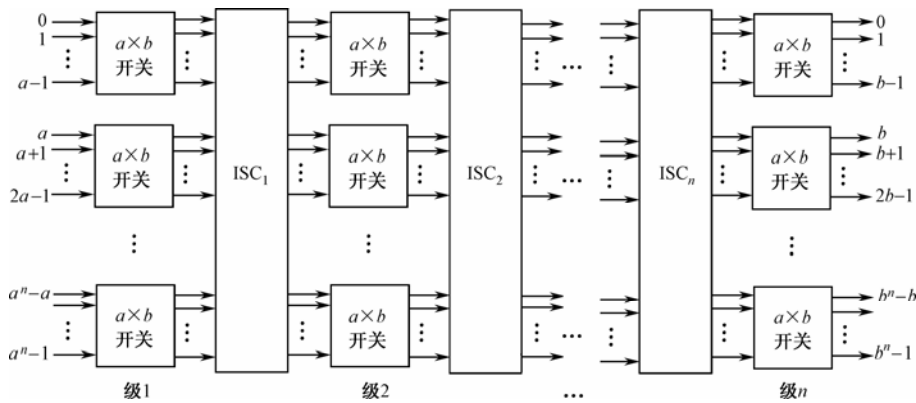


图 5-31 一种由 $a \times b$ 开关模块和级间连接模式 $ISC_1, ISC_2, \dots, ISC_n$ 构成的通用多级互连网络结构

各种 MIN 的区别就在于所用开关模块、控制方式和级间连接（ISC）模式的不同。最简单的开关模块是 2×2 开关，如果只有直送和交换，称为二功能交换单元；如果有直送、交换、上播和下播，称为四功能交换单元（如图 5-29 所示）。控制方式是指对各开关模块进行控制的方式，一般有三种：

- （1）级控制。每一级的所有开关只用一个控制信号控制，同级开关只能处于相同状态。
- （2）单元控制。每一个开关都有自己的单独控制信号，各开关处于不同的状态。
- （3）部分级控制。第 i 级的所有开关分别用 $i+1$ 个信号控制， $0 \leq i \leq n-1, n$ 为级数。例如，第 0 级（ $i=0$ ），则用一个控制信号，该级所有开关处于同一状态；第 1 级（ $i=1$ ）则用两个控制信号，该级开关分为两部分，相同部分开关状态一样，而不同部分开关状态可以不同；其他级可以类推。

常用的级间连接模式有混洗、立方体、PM2I、蝶式、纵横交叉等。

按照 MIN 对输入与输出的连接程度不同，分为阻塞网络、非阻塞网络和可重排非阻塞网络。在同时实现两对或多对入端与出端之间连接时，可能会出现开关状态设置上的冲突，称为阻塞网络。反之，称为非阻塞网络或称为全排列网络。非阻塞网络的优点是连接的灵活性好，缺点是连线多、控制复杂和成本高。典型的阻塞网络有多级立方体网络、多级混洗交换网络（Omega 网络）、多级 PM2I 网络，基准网络等。典型的非阻塞网络有多级 CLOS 网络等。可重排非阻塞网络是指为一对空闲的输入/输出结点建立通路时，可能影响当前正在使用的配置，但通过重新设置，仍可实现原来的要求并建立新的一对结点的通路，典型的例子如 BENES 网络。

1. 多级立方体网络

多级立方体网络有 STARAN 网络和间接二进制 n 立方体网络。以 8 个处理单元（ $N=8$ ）为例，其结构如图 5-32 所示。两者的共同点是：第 i 级（ $0 \leq i \leq n-1$ ）开关处于交换状态时，实现的是 $Cube_i$ 互连函数，且都采用二功能交换单元。两者差别仅在于控制方式上：STARAN

采用级控制和部分级控制，而间接二进制 n 立方体网络用单元控制。

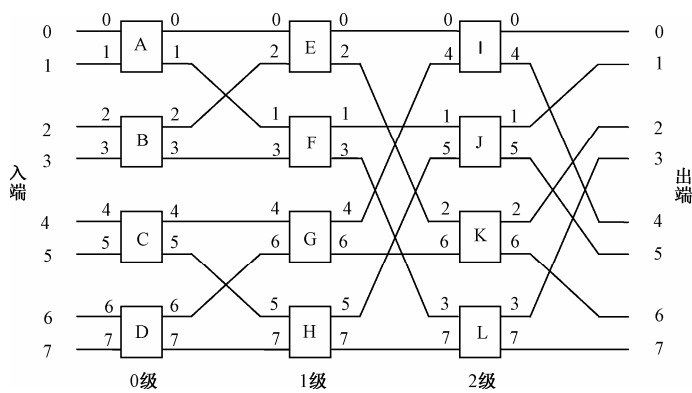


图 5-32 $N=8$ 的立方体多级互连网络

当 STARAN 网络用作交换网络，采用级控制。当有 N 个处理单元时，网络级数 $n=\log_2 N$ ，每级有 2^{n-1} 个二功能交换开关，同级用一个控制信号。当第 i 级控制信号为 0 时，该级开关都完成直送功能；当第 i 级控制信号为 1 时，该级开关都完成交换功能，实现 Cube_i 互连函数。例如， $N=8$ ， $n=\log_2 8=3$ ，这是一个有 8 个处理单元的三级 STARAN 网络，每级有 $2^{3-1}=4$ 个二功能交换开关，共有 $4 \times 3=12$ 个，每级用一个控制信号，共需要三个控制信号 $K_2K_1K_0$ ，级控制信号的组合及所实现的功能如表 5-4 所示。

表 5-4 STARAN 交换网络 ($N=8$) 级控制信号的组合及所实现的功能

		级控制信号 ($K_2K_1K_0$)							
		000	001	010	011	100	101	110	111
入 端 号	0	0	1	2	3	4	5	6	7
	1	1	0	3	2	5	4	7	6
	2	2	3	0	1	6	7	4	5
	3	3	2	1	0	7	6	5	4
	4	4	5	6	7	0	1	2	3
	5	5	4	7	6	1	0	3	2
	6	6	7	4	5	2	3	0	1
	7	7	6	5	4	3	2	1	0
换 函 数 功 能 执 行 的 交		恒等	四组 二元	四组二元+ 二组四元	二组四元	二组四元+ 一组八元	四组二元+二组四元+ 一组八元	四组二元+ 一组八元	一组八 元
		i	Cube_0	Cube_1	Cube_0+ Cube_1	Cube_2	$\text{Cube}_0+\text{Cube}_2$	Cube_1+ Cube_2	Cube_0+ Cube_1+ Cube_2

从表 5-4 可以看出，当 $K_2K_1K_0=001$ 时，0 级交换，1 级和 2 级直送，实现 Cube_0 互连函数。当 $K_2K_1K_0=011$ 时，各级开关状态如图 5-33 所示，实现 $\text{Cube}_0+\text{Cube}_1$ 功能，即二进制编码为 $P_2P_1P_0$ 的处理单元与二进制编码为 $P_2\bar{P}_1\bar{P}_0$ 的处理单元互连。图 5-33 给出了 0 号处理单元（二进制编码 000）到 3 号处理单元（二进制编码 011）的路由选择过程。

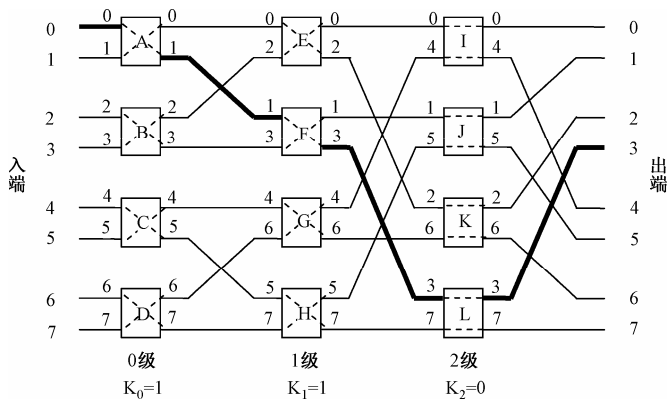


图 5-33 当级控制信号 $K_2K_1K_0=011$ 时各级交换开关状态图

在表 5-4 中，当 $K_2K_1K_0=101$ 时，各处理单元执行的交换函数功能次序是四组二元+二组四元+一组八元，其交换过程如表 5-5 所示。最终实现的是 $Cube_0 + Cube_2$ 功能。

表 5-5 $K_2K_1K_0=101$ 时的交换过程

入端排列	0	1	2	3	4	5	6	7
分成四组	0	1	2	3	4	5	6	7
每组二元交换	1	0	3	2	5	4	7	6
分成二组	1	0	3	2	5	4	7	6
每组四元交换	2	3	0	1	6	7	4	5
分成一组	2	3	0	1	6	7	4	5
每组八元交换	5	4	7	6	1	0	3	2

注：表中 | 表示分组。

当 STARAN 网络采用部分级控制时，实现的是移数网络。第 i 级的 2^{n-1} 个开关分成 $i+1$ 组，每组用一个控制信号控制。对于 n 级 STARAN 移数网络，从第 0 级到第 $n-1$ 级所需控制信号个数分别为 1, 2, 3, ..., n 个。当处理单元数 $N=8$ 时，有 $n=3$ 级，每级控制信号个数分别为 1, 2, 3，共 6 个，其每级控制信号分组和控制结果如表 5-6 所示。

STARAN 移数网络的置换函数可以表示为

$$\alpha(x)=(x+2^m) \mod 2^p$$

式中， p 和 m 为整数，且 $0 \leq m < p \leq n$ 。

例如，表 5-6 中功能部分的第 1 列，当控制信号 $A=B=C=D=0$ 、 $F=H=0$ 、 $E=G=1$ 、 $I=1$ 、 $J=0$ 、 $K=L=0$ 时，实现的功能是移 1 模 8，即 $m=0$ ($2^m=1$)、 $P=3$ ($2^P=8$)。移数网络将入端号 0~7 分一组，经网络内开关交换后，入端号 0 接出端号 1，入端号 1 接出端号 2，依次类推，将入端号循环移动一位，作为出端号。又如，表 5-6 中功能部分第 5 列，当控制信号 $A=B=C=D=0$ 、 $E=G=1$ 、 $F=H=1$ 、 $I=0$ 、 $J=0$ 、 $K=L=0$ 时，实现的功能是移 2 模 4，即 $m=1$ ($2^m=2$)， $P=2$ ($2^P=4$)。移数网络将入端号分成二组：0~3 为一组，4~7 为另一组，经网络内开关交换后，入端号 0 接出端号 2，入端号 1 接出端号 3，入端号 2 接出端号 0，入端号 3 接出端号 1，将入端号 0~3 循环移动两位作为出端号。另一组入端号 4~7 类似处理。

表 5-6 三级移数网络能实现的入/出端连接及移数函数功能

部分级控制信号	2 级	K, L	0	0	1	0	0	0	0
		J	0	1	1	0	0	0	0
		I	1	1	1	0	0	0	0
	1 级	F, H	0	1	0	0	1	0	0
		E, G	1	1	0	1	1	0	0
	0 级	A, B, C, D	0	0	0	1	0	1	0
入端号		0	1	2	4	1	2	1	0
		1	2	3	5	2	3	0	1
		2	3	4	6	3	0	3	2
		3	4	5	7	0	1	2	3
		4	5	6	0	5	6	5	4
		5	6	7	1	6	7	4	5
		6	7	0	2	7	4	7	6
		7	0	1	3	4	5	6	7
相当于实现的移数功能			移 1 mod 8	移 2 mod 8	移 4 mod 8	移 1 mod 4	移 2 mod 4	移 1 mod 2	不移 全等

2. 多级混洗交换网络

多级混洗交换网络又称 Omega 网，由 n 级相同的网络组成，每一级由全混洗（Shuffle）拓扑和 2^{n-1} 个四功能交换单元构成，采用单元控制方式。一个有 N 个处理单元的 Omega 网络需要 $n=\log_2N$ 级，每级有 2^{n-1} 个采用单元控制的 2×2 四功能交换开关。 $N=8$ 的多级混洗交换网络如图 5-34 所示。

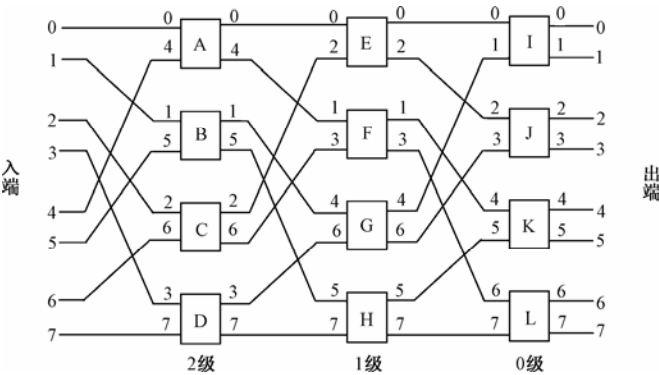


图 5-34 $N=8$ 的多级混洗交换网络

如果 Omega 网采用级控制，并且开关只有直送和交换两种功能，则它称为 STARAN 网络的逆网络，即它们的输入端和输出端的数据流向相反。将图 5-32 中的 F 和 G 两个交换开关连同它们的输入、输出端一起互换位置即可，如图 5-35（a）所示。图 5-35（b）是 Omega 网络，两者正好互逆。

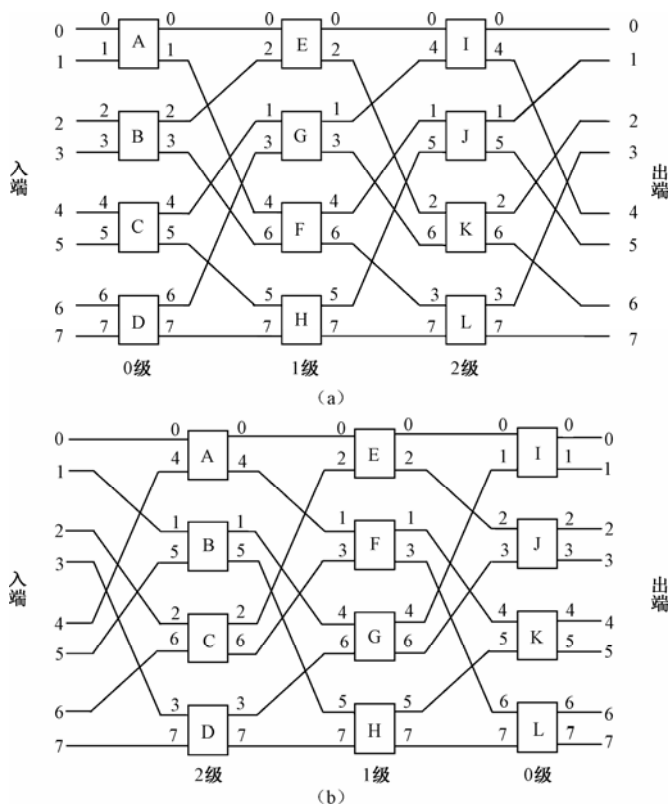


图 5-35 $N=8$ 的多级立方体网络和 Omega 网络的关系

3. 多级PM2I网络

多级 PM2I 网络有 n 级单元间连接，每一级都是把前后两列各 $N=2^n$ 个处理单元按 PM2I 拓扑互连。 $N=8$ 的多级 PM2I 网络如图 5-36 所示。对于第 i 级而言 ($0 \leq i \leq n-1$ ，图 5-36 中 $0 \leq i \leq 2$)，每个输入端 j ($0 \leq j \leq N-1$) 都有三根线分别连接到输出端 $(j-2^i) \bmod N$ (图中以虚线表示)、 j (图中以实线表示) 和 $(j+2^i) \bmod N$ (图中以粗实线表示)。第 0 级实现 $PM2_{\pm 0}$ ，第 1 级实现 $PM2_{\pm 1}$ ，第 2 级实现 $PM2_{\pm 2}$ 。单级 PM2I 网络最大距离为 $\lceil n/2 \rceil$ ，但组成多级 PM2I 网络时用了 $n=\log_2 N$ 级，图 5-36 $N=8$ 多级 PM2I 网络用了 $n=3$ 级，因此从一个处理单元到另一个处理单元的路径有多条可供选择。例如实现 6 号处理单元与 4 号处理单元互连，可以选择 $6 \rightarrow 6 \rightarrow 4 \rightarrow 4$ 或 $6 \rightarrow 2 \rightarrow 4 \rightarrow 4$ 等多条路径完成。多级 PM2I 网络不使用交换开关，可以提供冗余通路，这有利于提高系统的可靠性。

当多级 PM2I 网络的各级处理单元均采用单元控制时，称为强化数据变换网络，简称 ADM (Augmented Data Manipulator) 网络。ADM 网络互连方式较灵活，但硬件结构复杂，成本较高。表 5-7 根据交换开关、控制方式和拓扑结构对上述 5 种互连网络进行了比较。需要注意的是，三种二进制立方体网络入端在左，出端在右，其级号依次为 0, 1, 2, \dots , $n-1$ ；而 Omega 网络和 ADM 网络入端在左，出端在右，其级号依次为 $n-1$, \dots , 2, 1, 0。从互连灵活性由低到高的次序是 STARAN 网络、间接二进制 n 方体网络、Omega 网络、ADM 网络，而复杂性和成本也依次增大。

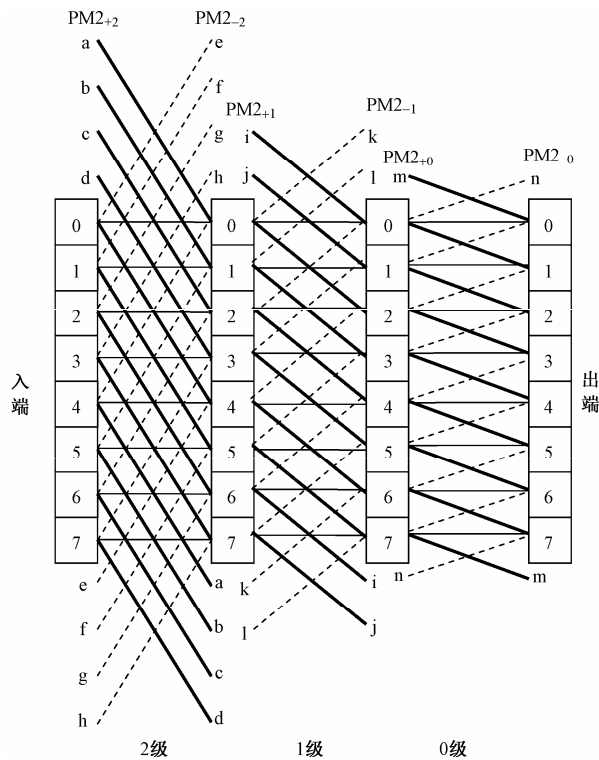


图 5-36 $N=8$ 的多级 PM2I 网络

表 5-7 5 种多级互连网络的比较

多级互连网络的三个参量 多级互连网络的类型		交换开关	控制方式	拓扑结构
二进制立方体网络	STARAN 交换网络	二功能交换单元	级控制	单级立方体网络
	STARAN 移数网络	二功能交换单元	部分级控制	单级立方体网络
	间接二进制 n 立方体网络	二功能交换单元	单元控制	单级立方体网络
多级混洗交换网络 (Omega 网络)		四功能交换单元	单元控制	单级混洗交换网络
强化数据变换网络 (ADM 网络)		多功能交换单元	单元控制	单级 PM2I 网络

4. 基准网络

基准网络常用于多级互连网络研发过程中，作为中间介质，模拟一种网络的拓扑和功能。图 5-37 为 $N=8$ 的基准网络的互连结构，从输入到输出的级间互连依次为恒等、逆 Shuffle、子逆 Shuffle 和恒等置换，网络中所有开关均为二功能交换单元，采用单元控制方式。

5. 多级CLOS网络

多级 CLOS 网络是一个非阻塞网络，三级 CLOS 交叉开关网络典型互连结构如图 5-38 所示。该网络输入和输出端口数都为 $n \times r$ 。输入级有 r 个交叉开关，每个交叉开关均为 $n \times m$ ；中间级有 m 个交叉开关，每个交叉开关均为 $r \times r$ ；输出级有 r 个交叉开关，每个交叉开关均为 $m \times n$ 。基准网络可以用三个参数 (m, n, r) 来描述，当 $m \geq 2n-1$ 时，多级 CLOS 网络 $N(m, n, r)$ 就是一个非阻塞网络。例如， $N(3, 2, 2)$ CLOS 网络，其互连如图 5-39 所示，每级有 12

个交叉点，共有 36 个交叉点。由于输入端和输出端各有 4 个，若使用单级 4×4 交叉开关，则一共只需要 16 个交叉点，所以本例使用单级交叉开关实现将更经济。多级 CLOS 网络所需的交叉点的个数为

$$C=r(m\times n)+m(r\times r)+r(m\times n)=mr(2n+r)$$

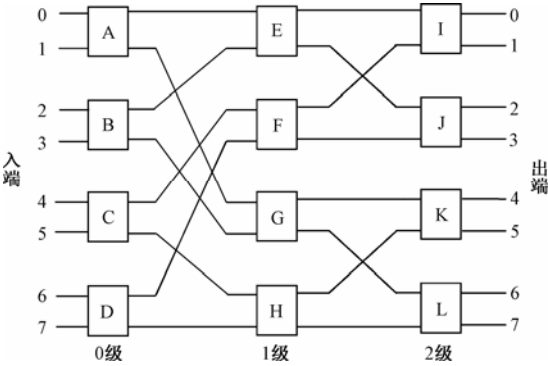


图 5-37 N=8 的基准网络互连

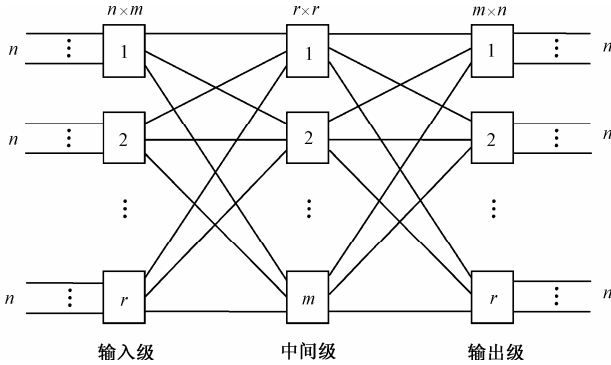


图 5-38 三级 CLOS 交叉开关网络互连

若直接用单级交叉开关实现，总共需要 $n\times n=n^2$ 个交叉点。当 $mr(2n+r)<n^2$ 时，即当 n 的值相当大时，选用 CLOS 网络不仅可以实现无阻塞互连，而且成本较低。由于 CLOS 网络由若干较小规模交叉开关组成，在工程上也较容易实现。

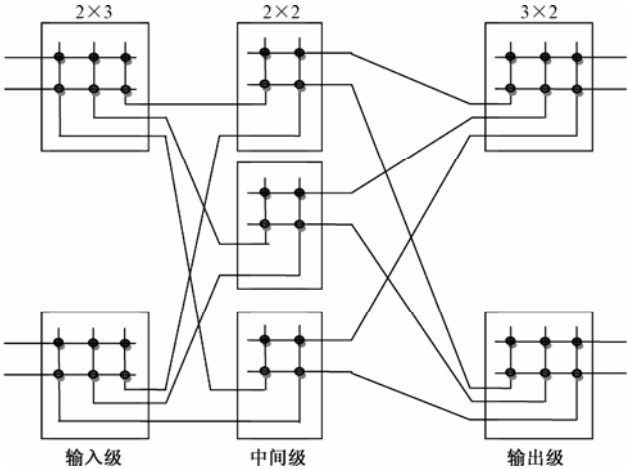


图 5-39 N(3, 2, 2)CLOS 交叉开关网络互连

6. 多级BENES可重排网络

将基准网络和它的逆网络连在一起，就组成了一个 BENES 网络。 $N=8$ 的 BENES 可重排网络如图 5-40 所示。图中用虚线框画在一起的两级开关是完全重复的，故可以合并成一级。开关都是二功能交换单元，采用单元控制。如果要求一个网络是可重排的，以改善阻塞情况，那么这个网络至少应该有两个以上的通道能够满足同一对结点间互连的要求，即路由寻径的算法不是唯一的。BENES 网络的级数为 $2\log_2 N-1$ ，每级由 $N/2$ 个 2×2 开关组成。8 个处理单元编号为 0~7，全排列置换共有 $8!=40320$ 种，而 BENES 网络可实现的置换共有 $2^{20}=1048576$ 种，比全排列要多得多，所以 BENES 网络是一个可重排非阻塞网络。

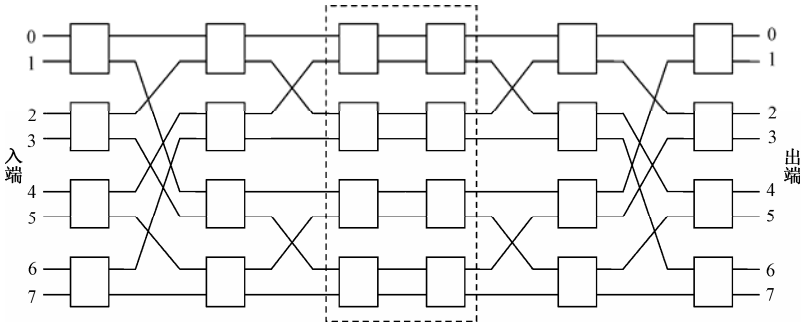


图 5-40 $N=8$ 的多级 BENES 可重排网络

7. 蝶式网络

多级蝶式网络是用交叉开关模块组成的。图 5-41 (a) 是 $N=64$ 的两级蝶式网络，每级由 8 个 8×8 交叉开关模块构成，级间采用 8 路 Shuffle 互连，互连函数为

$$\text{Shuffle}(P_5P_4P_3P_2P_1P_0)=P_2P_1P_0P_5P_4P_3$$

图 5-41 (b) 是有 512 个输入端的三级蝶式网络结构，其中第 0 级和第 1 级就是图 5-41 (a) 所示两级蝶式网络，共用 16 个 8×8 交叉开关。整个三级蝶式网络共用 192 个 8×8 交叉开关。若要构成更大的蝶式网络，只要增加级数即可。但蝶式网络不许广播连接，因此蝶式网络是 Omega 网络的子集。

8. 组合网络

当多个处理机同时访问某个存储器模块时，由于共享信号灯变量将其作为同步路障，就此形成一个热点 (the Not-spot Problem)，热点会大大降低网络的性能。所谓组合网络就是指在产生冲突的开关点上增加组合机构，把访问同一个目的结点的多个请求组合在一起。纽约大学的 Ultra Computer 和 IBM 公司的 RP3 多处理机在 Omega 网络上增加了组合机构，实现了在发生冲突的开关点上把访问同一目的结点的多个请求组合在一起。

组合网络使用“读—修改—写”原语 Fetch & Add(x, e) (或写成 FAA(x, e), Fetch-And-Add(x, e)), 可以并行地执行存储器修改操作。该原语对于原子存储器操作处理 N 路同步是很有效的，其复杂性与 N 无关。该原语中的 x 是共享存储器中的整型变量， e 是整型增量。当单台处理机执行该操作时，它的语义是：

```
Fetch & Add ( $x, e$ )
{temp  $\leftarrow x$ ;
 $x \leftarrow \text{temp}+e$ ;
return temp}
```

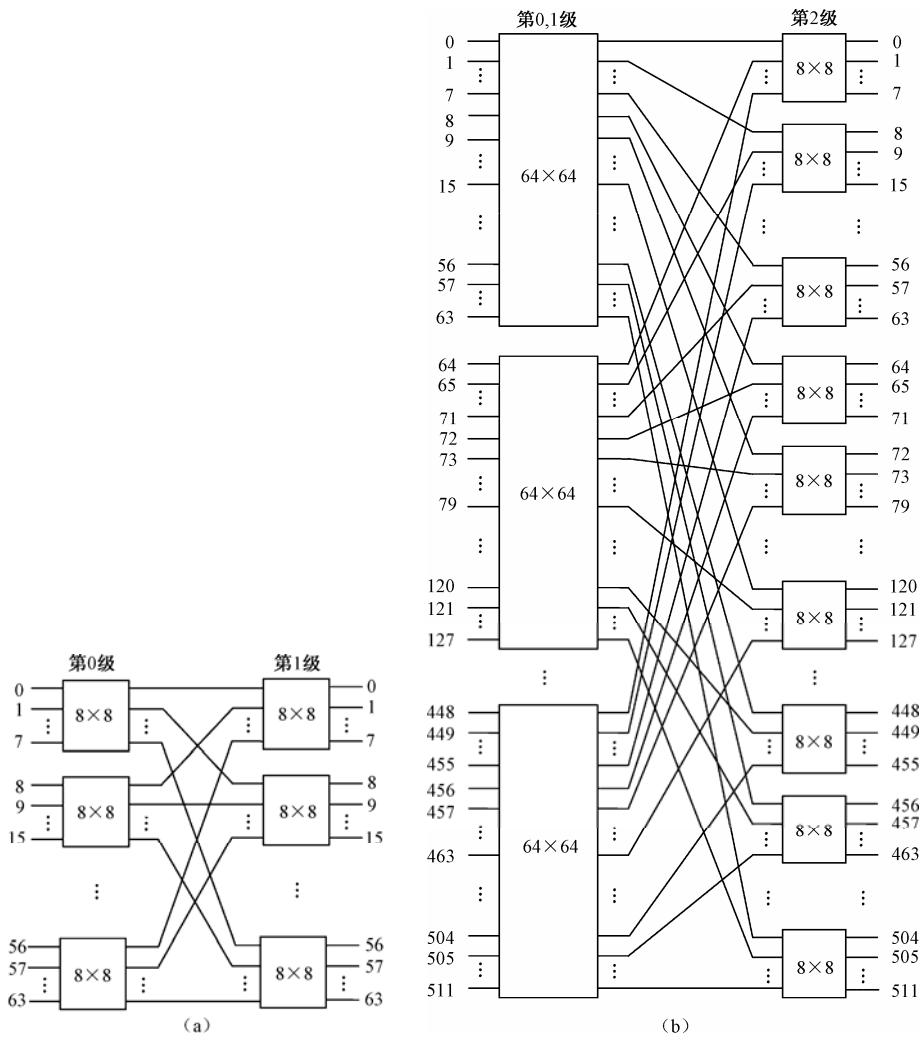



图 5-41 8×8 交叉开关构成模块结构的蝶式网络

当 N 台处理机同时对同一存储器地址执行 **Fetch & Add(x, e)**，按照串行化原则只能对存储器进行一次修改， N 个增量的和与存储器字 x 相加，产生新的值 $x+e_1+e_2+\cdots+e_N$ 。这个新值的产生与 N 个请求的串行顺序无关，即最后存储器内的值为 $x+e_1+e_2+\cdots+e_N$ 。但是，返回给每一个请求（共 N 个）的值都是不一样的，它取决于串行的顺序。在执行该操作时中间不允许中断。图 5-42 所示是两个 **Fetch & Add(x, e)** 操作组合成能同时访问共享变量的操作过程。其中 P_1 请求先执行， P_2 请求后执行。操作结束，存储器共享变量的值为 $x+e_1+e_2$ ，开关缓冲区内容为 e_1 ，返回 P_1 处理机的值是 x ，返回 P_2 处理机的值是 $x+e_1$ 。若 P_2 请求先执行， P_1 请求后执行，其过程类似，共享变量的值仍是 $x+e_1+e_2$ ，但开关缓冲区内容为 e_2 ， x 返回给 P_2 ， $x+e_2$ 返回给 P_1 。

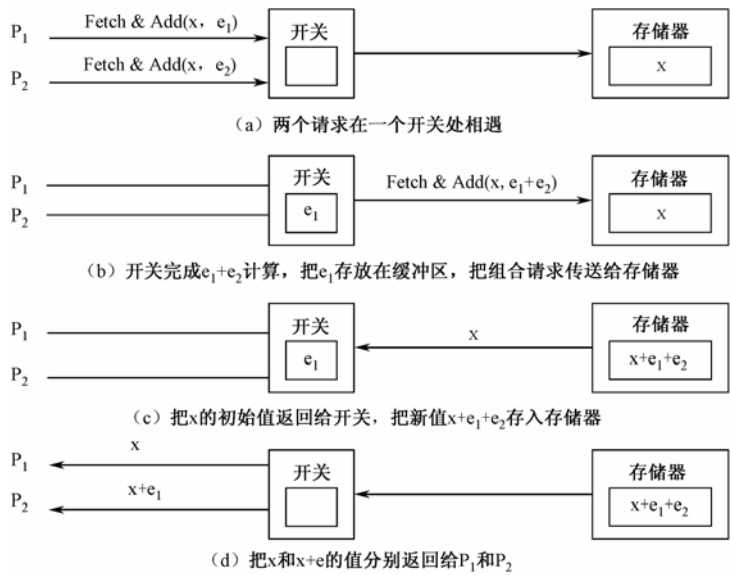


图 5-42 通过组合网络把两个 Fetch & Add 操作组合成能同时访问一个共享变量的操作

表 5-8 中汇总了动态网络的系统总线、多级网络、交叉开关的主要网络特性。系统总线的优点是成本最低, 缺点是每台处理机可用带宽较窄, 且容易产生故障, 有些容错系统采用双总线防止系统产生简单故障。交叉开关的优点是带宽和寻径性能最好, 缺点是硬件复杂性以 n^2 上升, 所以成本最高, 在网络规模较小时, 交叉开关是一种理想的选择。多级网络是二者之间的折中, 优点是采用模块结构, 可扩展性好, 缺点是时延随网络级数 $\log_2 n$ (n 为结点数) 增加而上升, 另外, 由于增加了连线和开关控制的复杂性, 成本也有所提升。

表 5-8 动态网络特性一览表

网 络 特 性	总 线 系 统	多 级 网 络	交 叉 开 关
单位数据传送的最小吋延	恒定	$O(\log_k n)$	恒定
每台处理机的带宽	$O(w/n) \sim O(w)$	$O(w) \sim O(nw)$	$O(w) \sim O(nw)$
连线复杂性	$O(w)$	$O(nw \log_k n)$	$O(n^2 w)$
开关复杂性	$O(n)$	$O(n \log_k n)$	$O(n^2)$
连接特性和寻径性能	一次只能一对一	只要网络不阻塞, 可实现某些置换和广播	全置换, 一次一个
典型计算机	Symmetry S1, Encore Multimax	BBNTC-2000 IBM RP3	CRAY Y-MP/816 Fujitsu VPP 500
评注	总线上假定有 n 台处理机: 总线宽度为 w 位	$n \times n$ MIN 采用 $k \times k$ 开关, 其线宽为 w 位	假定 $n \times n$ 交叉开关的线宽为 w 位

5.3.7 互连网络寻径

数据寻径用于结点间交换数据, 寻径网络可以是静态的或动态的。在多级网络情况下, 数据寻径是通过消息传递来实现的。硬件寻径器 (路由器) 可用于多个计算机结点间传送寻

径消息。网络的寻径功能强，可减少数据交换的时间，显著改善系统性能。常见数据寻径功能如下：

- （1）点对点（Point-to-Point）。仅有一个发送者和一个接收者，这是一对一通信。
 - （2）广播（Broadcast）和散射（Scatter）。广播是指一个进程（或称为根进程）向所有进程（包括自己）发送相同消息；散射是根进程对不同进程发送不同消息，是通用化的广播，广播和散射是一对多通信。
 - （3）汇合（Gather）和归约（Reduction）。汇合（也称聚集）是指根进程从每个进程处接收一个不同消息，共接收 n 个消息， n 是组的大小；归约是指根进程接收每个进程（包括自己）的局部值，然后在根进程中求和形成一个最后值。汇合和归约是多对一通信。
 - （4）置换（Permutation）。包括循环移动（Circular Shift）、扫描（Scan）、全交换（Total Exchange）等，多个进程中的每个进程只向一个进程发送或接收消息，置换是多对多通信。
- 常见的数据寻径方式如图 5-43 所示。

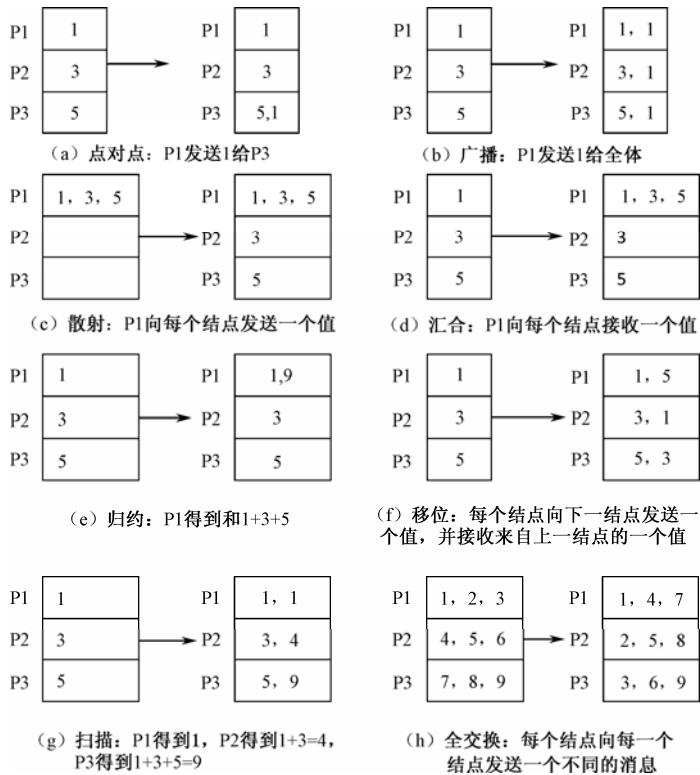


图 5-43 常见的数据寻径方式

寻径的目的是找出一条从源结点到目的结点的路径来传递消息。寻径分为确定和自适应两类。采用确定寻径时，路径完全由源结点地址和目的结点地址确定，与网络状况无关，寻找的路径是预先唯一确定的。自适应寻径与网络状况有关，可能会有多条路径。这两种寻址都需要无死锁算法。

1. 确定寻径

确定寻径（Deterministic Routing）基于维序概念。维序寻径概念是基于一种按照多维网

络维序的特定顺序来选择后继通道。在二维网格网络中称为 x - y 寻径，首先沿着 x 维方向确定路径，然后沿着 y 维方向选择路径。在超立方体（或 n 立方体）网络中，1977 年 Sullivan 和 Bashkow 提出了 E 立方体寻径（E-Cube Routing）方法。

（1）二维网格网络的 x - y 寻径。从任意源结点 $s=(x_1, y_1)$ 到任意目的结点 $d=(x_2, y_2)$ 寻径方向是从 s 开始，沿 x 方向寻径，一直到 d 所在的第 x_2 列为止，然后沿 y 方向直到 d 。 x - y 寻径路径方向共有 4 种模式：东-北，东-南，西-北和西-南。图 5-44 所示是二维网格网络连接 64 个结点（处理机）的 x - y 寻径，图中有 4 个(源，目的)对，用于说明二维网格网络的 4 种寻址模式。从结点(2, 1)到结点(7, 6)是走东-北路径，从结点(0, 7)到结点(4, 2)是走东-南路径，从结点(5, 4)到结点(2, 0)是走西-南路径，从结点(6, 3)到结点(1, 5)是走西-北路径。如果总是先沿 x 方向寻径，然后再沿 y 方向寻径，就不会出现死锁或循环等待现象。

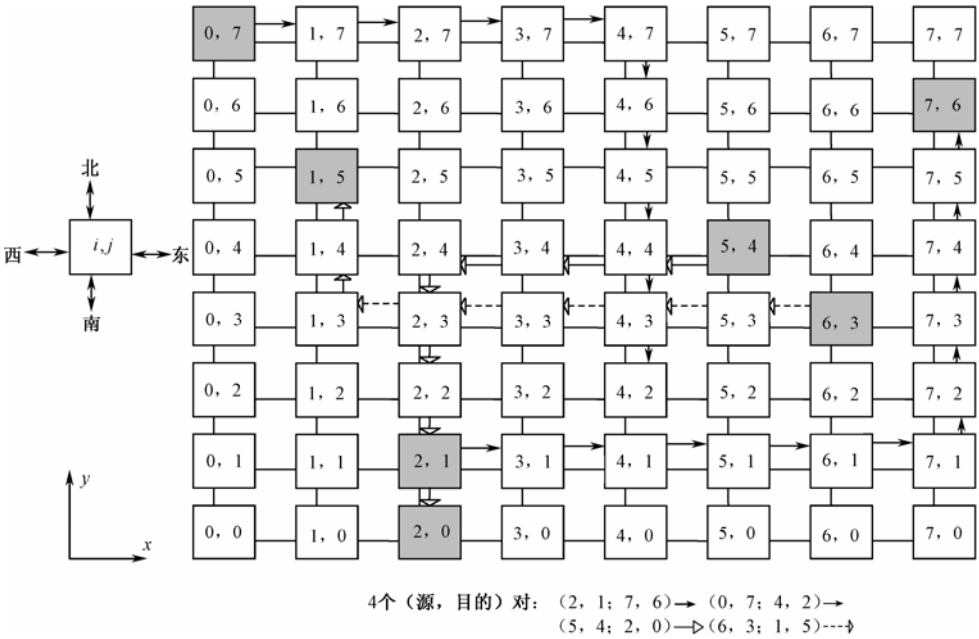


图 5-44 8×8=64 个结点二维网格网络的 x - y 寻径

（2）超立方体网络的 E 立方体寻径。设有一个 $N=2^n$ 个结点的 n 方体，每个结点的二进制编码 $b=b_{n-1}b_{n-2}\cdots b_1b_0$ ，这样，源结点二进制编码为 $s=s_{n-1}s_{n-2}\cdots s_1s_0$ ，目的结点二进制编码为 $d=d_{n-1}d_{n-2}\cdots d_1d_0$ 。现在要寻找一条从 s 到 d 的步数最小的路径。具体步骤如下：

- ① 将 n 维表示成 $i=1, 2, \cdots, n$ ，其中第 i 维对应于结点地址中的第 $i+1$ 位。设 $v=v_{n-1}v_{n-2}\cdots v_1v_0$ 是路径中的任一个结点。
- ② 计算方向位 $r=s \oplus d=r_nr_{n-1}\cdots r_2r_1$ 。
- ③ 从源结点 s 出发， $v=s$ ，然后从 r_1 起始，逐位判断：若 $r_1=1$ ，则 $v \oplus 2^{i-1}$ ，得到新的 v ；若 $r_1=0$ ，则跳过这一步，即维持原来的 v 。
- ④ $i+1 \rightarrow i$ ，依③判断下一个 r_i ，产生新的 v 或跳过，直至 $i=n$ 为止。最后的 $v=d$ ，即最后的 v 必为 d 。

例如，有四维超立方体如图 5-45 所示， $n=4$ ，现从源结点 $s=0110$ 出发，寻径到目的结点 $d=1101$ ，过程如下：

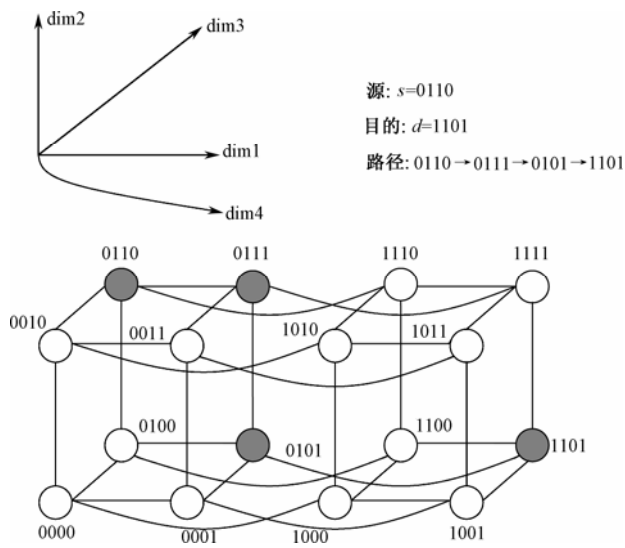


图 5-45 16 个结点四维超立方体网络的 E 方体寻径

- ① 计算方向位 $r = s \oplus d = 0110 \oplus 1101 = 1011 = r_4 r_3 r_2 r_1$ 。
- ② 从源结点 s 出发, $v = s = 0110$, 判 $r_1 = 1$, 则 $v \oplus 2^0 = 0110 \oplus 0001 = 0111$, 得到下一个结点 $v = 0111$ 。
- ③ 判断 $r_2 = 1$, 则 $v \oplus 2^1 = 0111 \oplus 0010 = 0101$, 得到新结点 $v = 0101$ 。
- ④ 判断 $r_3 = 0$, 则跳过。
- ⑤ 判断 $r_4 = 1$, 则 $v \oplus 2^3 = 0101 \oplus 1000 = 1101$, 得到新结点 $v = 1101$, 即 $v = d = 1101$, 到达目的结点。

所以, 寻径结果是: $0110 \rightarrow 0111 \rightarrow 0101 \rightarrow 1101$ 。

2. 自适应寻径

自适应寻径通常采用虚拟通道, 使寻径更经济、更灵活, 但是自适应寻径要特别注意避免死锁。图 5-46 是一个用 x - y 寻径的网格网络, 在 y 方向采用了 2 对虚拟通道, 图 5-46 (c) 的虚拟网络可以避免消息在向西传输时出现死锁, 因为所有向东的 x 通道都没有使用。同样地, 图 5-46 (d) 的虚拟网络使用另一组 y 方向的虚拟通道支持向东的传输。在不同时刻使用两个虚拟网络, 就可以自动避免死锁。

图 5-47 是在 x 方向和 y 方向各有两条虚拟通道的网格网络, 可以生成 4 个虚拟网络, 如图 5-47 (b), (c), (d), (e) 所示。任何一个虚拟网络都不会出现环路, 因此在这些网络上实现 x - y 寻径方法时, 完全可以避免死锁。如果相邻结点之间的两对通道都是物理通道, 那么 4 个虚拟网络中任意两个都可以同时使用而不会产生冲突。如果相邻结点之间双虚拟通道共同使用一对物理通道, 那么只有图 5-47 (b) 和 (e) 或 (c) 和 (d) 可以同时使用。其他组合如 (b) 和 (c) (在 y 方向冲突)、或 (b) 和 (d) (在 x 方向冲突), 或 (d) 和 (e) (在 y 方向冲突), 或 (c) 和 (e) (在 x 方向冲突) 都不能同时使用, 因为缺少物理通道。如果增加网络通道数, 寻径的自适应性也会增加, 但成本也将大大增加。

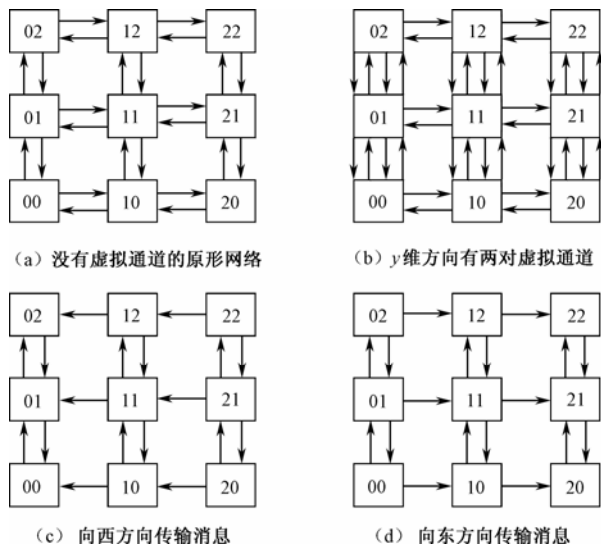


图 5-46 利用虚拟通道避免死锁的自适应 x - y 寻径

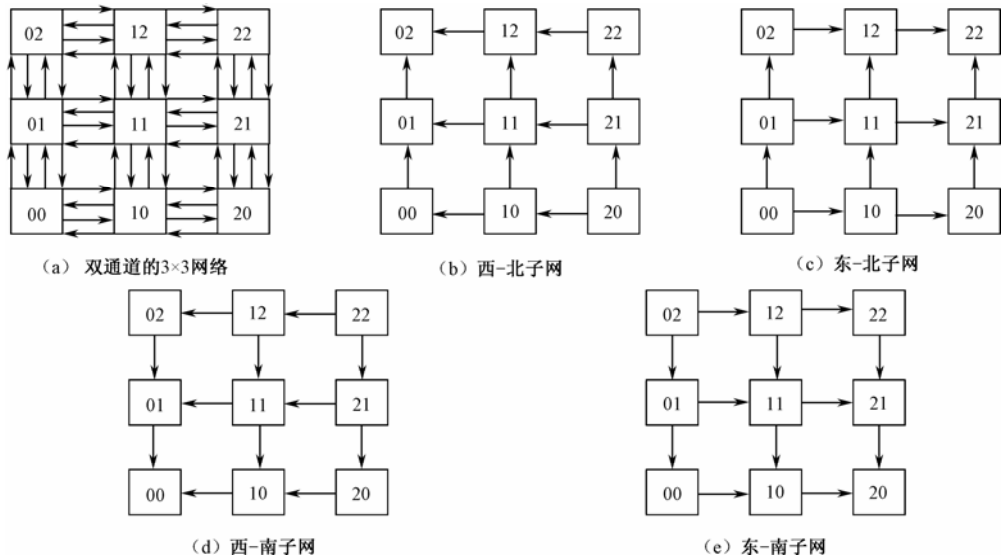


图 5-47 双通道网格网络可实现 4 个虚拟网络

互连网络也可以如图 5-48 所示进行分类。

(1) 共享介质网络。该网络有总线形和环形两类，其共同点是在同一时刻都只允许一个主控模块进行存取。总线形包括底板总线（又分为单总线，如 SGIPOWERp_{th}-2，DEC LSB；以及双总线，如 SUN XD Bus）、争用总线（如以太网 Ethernet）和令牌总线（Token Bus，如 ARCnet）。环形有 FDDI 环和 IBM Token Ring。

(2) 非阻塞网络。该网络是逻辑交叉开关网络，只要无不同输入端口同时向一个输出端口发送消息，则消息通信不会阻塞。用物理的交叉开关和空分的总线都可以设计无阻塞网络。由于交叉开关的硬件复杂性以 N^2 上升，所以它造价昂贵，但是交叉开关的带宽和寻径性能最好。无阻塞网络主要用于组成小规模的高性能互连网络，一般从 4 个端口到 32 个端口。无阻

塞网络包括二维网络的交换开关网络（如 Cray x/y—MP, DEC GIGAswitch, Myrinet）、共享存储器网络（如 CNET Prelude）、空分总线（如 Fore ATMASX-100）和 CLOS 网络。

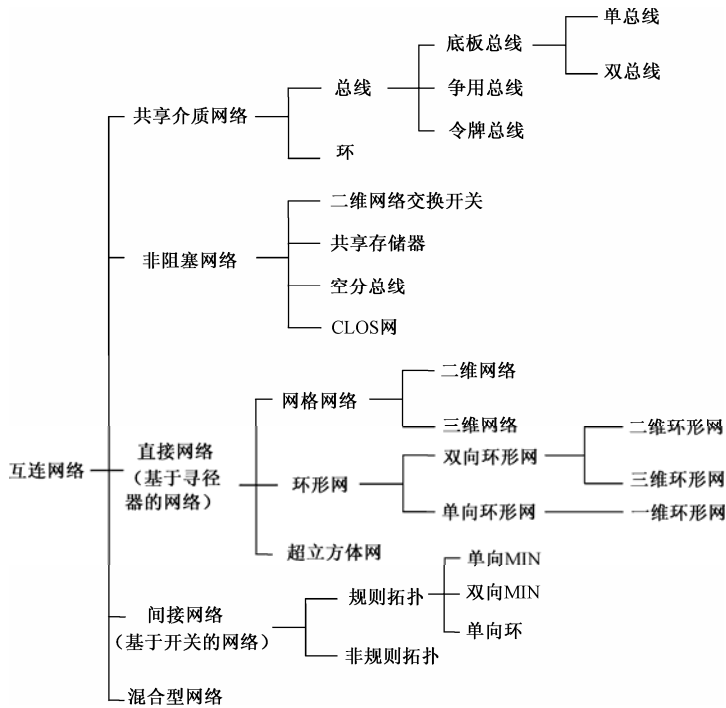


图 5-48 互连网络分类

（3）直接网络。该网络中处理机是点到点连接的，也称静态网络。其所有结点都有一个寻径器用于消息通信，故称为基于寻径器的网络。寻径器通过内部通道与本地处理机连接，通过外部通道与相邻结点连接，外部通道决定了互连网络拓扑结构。直接网络分为三类：网格、环形网和超立方体。网格又分为二维网格（如 Intel Paragon）和三维网格（如 MIT J machine）。环形网分为单向一维环形网（如 KSR）和双向环形网，双向环形网有二维的（如 Intel/CMV iWARP）和三维的（如 Cray T3D）。超立方体的典型机器有 Intel iPSC 和 nCUBE。

（4）间接网络。也称基于开关的网络，它适合于大规模互连网络。每一个结点有一个网络适配器连接到网络开关上，其互连方式决定了网络的拓扑，大多数是基于各种规则的多级互连网络的拓扑结构。间接网络可以分为规则拓扑和非规则拓扑。规则拓扑又分为单向多级网络（如 NEC Cenju-3）、双向多级网络（如 IBM SP, TMC CM-5, Meiko CS-2）和单向环（如 KSR-2-level ring, Conver Exemplar）。

（5）混合网络。指互连网络中混合了上述多种网络。例如，互连网络的主干网络是超立方体，而每个结点是网格网络。在 Conver Exemplar 中，每个结点是 8 个处理器的交叉开关网络，而所有结点用 4 个重复的单向环互连。

5.4 阵列处理机

并行处理机（Parallel Processor）也称阵列处理机（Array Processor）。通过重复设置大量

相同的处理单元 PE (Processing Element)，将它们按一定方式互连成阵列，在单一控制部件 CU (Control Unit) 控制下，阵列内各个 PE 对各自所分配的不同数据并行执行同一条指令规定的操作。因此，阵列机是操作级并行的 SIMD 计算机。PE 的核心是算术逻辑运算部件，按地址访问随机存储器。这种处理机主要用于对向量和数组进行高速运算的场合。

5.4.1 阵列处理机结构

并行处理机由于存储器的组成方式不同，有两种不同的基本架构：分布式存储器的并行处理机结构（如图 5-3 所示）和共享式存储器的并行处理机结构（如图 5-4 所示）。采用分布式存储器结构的并行处理机是 SIMD 的主流。典型代表有美国 Illianos 大学于 20 世纪 60 年代开始研制，1972 年由 Burroughs 公司生产的 ILLIAC IV 阵列处理机，美国 Goodyear 宇航公司 1979 年研制成功的巨型并行处理机 MPP (Massively Parallel Processor)，英国 ICL 公司 1974 年开始设计、1980 年生产的分布式阵列处理机 DAP (Distributed Array Processor)，美国 Thinking Machines 公司的连接机 (Connection Machine) CM-2，MasPar 公司的 MP-1，Active Memory Technology 公司的 DAP 600 系列机等。采用共享式存储器结构的并行处理机的典型例子有 Illianos 大学和 Burroughs 公司联合研制的科学处理机 BSP (Burroughs Scientific Processor)。SIMD 计算机的发展过程如图 5-49 所示。ILLIAC IV 是阵列机的代表。

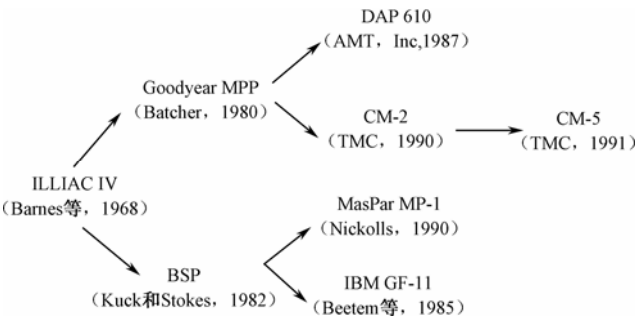


图 5-49 SIMD 计算机发展过程

ILLIAC IV 系统原理总框图如图 5-50 所示。它是由两大部分组成：即 ILLIAC IV 阵列和 ILLIAC IV 输入/输出系统。整个机器是由三种类型处理机组成的多机系统：一是实现向量、数组运算的处理单元阵列 (Processing Element Array)；二是阵列控制器 (Array Control Unit)，它既是处理单元阵列的控制部分，又是一台相对独立的小型标量处理机；三是一台 Burroughs B6700 计算机，作为前台机，担负 ILLIAC IV 输入/输出系统和操作系统管理功能。

1. ILLIAC IV 阵列

ILLIAC IV 阵列由 64 个处理单元、64 个处理单元存储器和存储器逻辑部件组成，属于分布式存储器结构。64 个处理部件 $PU_0 \sim PU_{63}$ 排列成 8×8 的方阵，每个 PU_i 只和其左、右、上、下 4 个邻近 $PU_{i+1} \pmod{64}$ 、 $PU_{i-1} \pmod{64}$ 、 $PU_{i+8} \pmod{64}$ 和 $PU_{i-8} \pmod{64}$ 相连，从而构成闭合螺旋线阵列，如图 5-51 所示。任意两个处理单元间通信可以用软件方法寻径。 $n \times n$ 个单元组成的阵列中，任意两个处理单元之间最短距离小于等于 $n-1$ 步。ILLIAC IV 阵列最短距离不会超过 7 步。例如，从 PU_{10} 到 PU_{46} ，其路径为 $PU_{10} \rightarrow PU_9 \rightarrow PU_8 \rightarrow PU_0 \rightarrow PU_{63} \rightarrow PU_{62} \rightarrow PU_{54} \rightarrow PU_{46}$ 。

器兼互连寄存器，完成 4 个方向的 PU 之间数据直接交流；④ RGS 是通用寄存器，暂存中间结果。这 4 个寄存器都是 64 位的。⑤ RGX 是变址寄存器，16 位，通过地址加法器形成有效地址，经过存储器地址寄存器 MAR 送存储器逻辑部件 MLU。⑥ RGM 是模块寄存器，8 位，它的 E 和 E1 位是“活动”标志位，F 和 F1 位是溢出位（上溢和下溢）标志位，G，H，I，J 位保存测试结果。RGM 在 CU 监督下，一旦出错（溢出）就向 CU 发出陷阱中断。E 和 E1 用于控制 RGA，RGS 和阵列存储器工作，E 还控制 RGX。当 PU 进行 32 位字长运算时，E 和 E1 相互独立。活动标志位可对每个 PU 进行单独控制，只有处于活动的 PU 才执行单指令流规定的共同操作。RGM 可由程序设置成“活动”或“不活动”。其他部件是：加/乘算术单元 AU、逻辑单元 LU、移位单元 SU 和地址加法器 ADA 等。PU 可对 64 位、32 位和 8 位操作数进行多种算术和逻辑运算，定点运算字长 48 位、24 位和 8 位。因此，阵列中 64 个 PU 可视作 64 个（64 位）、128 个（32 位）或 512 个（8 位）处理单元在发挥作用。并行加法速度如下：8 位定点加法是 10000MIPS，64 位浮点加法是 150MFLOPS。

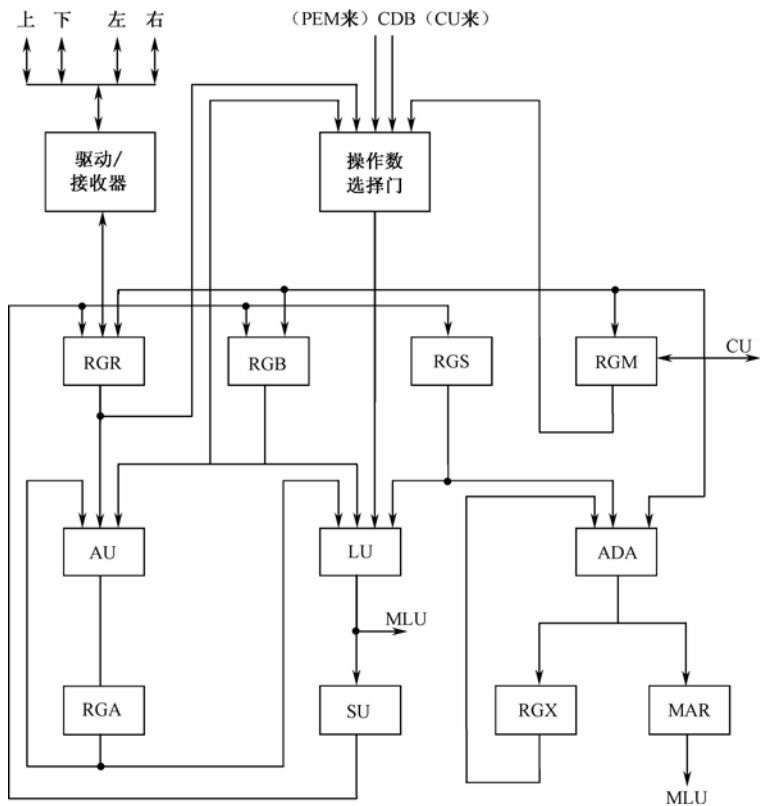


图 5-52 ILLIAC IV 处理单元 PU 原理框图

每个 PU 都有本地存储器 PEM，容量为 $2K \times 64$ ，存取时间 $\leq 350ns$ 。64 个 PEM 构成阵列 MEM，存放数据和指令。整个阵列 MEM 受 CU 控制，每个 PU 只能访问自己的 PEM。公共数据由 CU 经过公共数据总线广播到 64 个 PU 中，这样不但节省存储空间，而且允许公共数据存取与其他操作在时间上重叠。阵列 MEM 如同一个二维访问存储器，若把 64 个 PEM 看成行，每一个 PEM 本身看成行，则 CU 是按列访问，PU 是按行访问。阵列 MEM 采用双重变址，CU 实现所有 PU 的公共变址，每个 PU 内部可以单独变址，最后的有效地址

$EA=a+(b)+(c_i)$ 。其中， a 是指令地址， (b) 是CU中央变址寄存器内容， (c_i) 是 PU_i 局部变址寄存器内容，这增加了灵活性。PU与PEM经过存储逻辑部件MLU相连，MLU有信息寄存器、控制逻辑等，可实现PEM与PU，CU和I/O之间信息传递。

2. 阵列控制器

阵列控制器CU除了对阵列处理单元PU实现控制以外，还用本身的指令系统完成标量操作，在时间上与各PU的数组操作重叠起来。CU与PU阵列之间的连接如图5-50所示（ PU_i 由 PE_i 和 PEM_i 组成），共有4条信息通路：

（1）CU总线。PU内存储器PEM经过CU总线把指令和数据送往CU，以 $8 \times 64b$ 为一个信息块传送。指令是指分布存放在PEM内的用户程序的指令；而数据是处理的公共数据，先将公共数据送到CU，再利用CU的广播功能送到其他各个PU。

（2）公共数据总线CDB（Common Data Bus）。CDB是64位总线，作为向64个PU同时广播公共数据的通路。例如，公共乘数作为一个常数就不必在64个PEM中重复存放，可以由CU某个寄存器经CDB传送至各个PU。CDB也传送指令的操作数和地址。

（3）模式位线（Mode Bit Line）。每个PU经过模式位线把它的模式寄存器状态送CU，其中包括该PU的“活动”状态位。只有位于“活动”状态的PU才执行单指令流所规定的公共操作。64个PU送往CU的模式位在CU的累加寄存器中拼成一个模式字，供CU执行测试指令时使用，并根据测试结果控制程序转移。

（4）指令控制线。CU经过约200根指令控制线将处理单元微操作控制信号和处理单元存储器地址、读/写控制信号送到阵列处理单元PE和存储器逻辑部件MLU，实现控制功能。

归纳起来，CU主要功能有以下5个方面：

（1）对单指令流进行控制和译码，并执行标量操作指令。

（2）向各个PU发出执行数组操作指令所需的控制信号。

（3）产生并向所有PU广播公共的地址部分。

（4）产生并向所有PU广播公共的数据。

（5）接收和处理由各个PE在计算出错时发出的、系统I/O操作时发出的以及B6700产生的陷阱中断。

3. 输入/输出系统

ILLIAC IV输入/输出系统由磁盘文件系统DFS、I/O分系统和B6700组成。而I/O分系统又由输入/输出开关IOS、控制描述字控制器CDC和输入/输出缓冲存储器BIOM三部分组成。

DFS是两套大容量并行读/写磁盘系统及其相应的控制器，总容量为125MB，最大传输速率为62.75MB/s，平均等待时间为19.6ms。如果两个磁盘系统同时发送或接收数据，最大传输速率可达125MB/s。

I/O分系统内IOS功能有两个：一是把DFS或实时装置转接到阵列存储器，进行大批数据I/O传送；二是作为DFS和PEM之间的缓冲。CDC的功能是对CU的I/O请求进行管理：CDC向B6700发中断；B6700响应，并通过CDC向CU送回响应代码，在CU中设置状态控制字；然后CDC使B6700启动PEM加载过程，由DFS向PEM送入程序和数据，加载结束由CDC向CU发控制信号，CU开始执行ILLIAC IV的程序。BIOM处在DFS和B6700之间，实现两者之间传送频带匹配。B6700频带是10MB/s，DFS频带是62.5MB/s，两者相差6倍之多。BIOM作为缓冲，把B6700的48b/字变换为ILLIAC IV的64b/字，同时以两个

字共 128b 数据宽度送 DFS。BIOM 由 4 个 PE 存储器组成, 容量为 64KB。

B6700 由 CPU (另一 CPU 可选)、32K×48 (即 192KB) 内存 (可扩充至 3072KB)、经过多路开关控制的外设 (其中包括一台 125GB 激光外存) 以及 ARPA 网络接口等组成。B6700 的作用是管理整个系统的资源, 完成用户程序的编译或汇编, 为 ILLIAC IV 进行作业调度、存储分配、产生 I/O 控制字送 CDC、响应和处理中断, 以及提供操作系统其他服务功能。因此, B6700 是 ILLIAC IV 系统的前台机, 直接面向用户, 而 CU 和 PU 阵列构成后台机, 完成数组或标量的计算和处理, 实现并行处理功能。

5.4.2 阵列处理机算法

以下列举在 ILLIAC IV 上采用的算法, 说明一般阵列机的工作原理。

(1) 有限差分问题。阵列机的二维阵列结构特别适合计算在网络上定义的有限差分函数。先假定作为初始条件的网络边缘点的函数值是已知的, 而内部各点函数值为零, 并把这些内部网格点分配给各个处理单元。然后根据有限差分公式多次迭代求值, 直至连续两次迭代所求值相差很小为止。当然必须已知迭代过程是收敛的。

当求解问题的内部网格点数大于处理单元个数时, 把网格点划分成多个子网格点, 然后由阵列机分批运行。

(2) 矩阵问题。矩阵运算是最适合阵列机运行的。如 A , B 两个矩阵相加, 只要把 A 和 B 居于相应位置的一对分量存放在同一个处理单元存储器内即可。当阵列机执行加法公共操作时, 每个处理单元都将处于本结点的 A_i 和 B_i 两个矩阵元素进行加法运算, 其和即为矩阵和的对应元素。

(3) 矩阵乘问题。矩阵乘是二维数组。设 A , B , C 为三个 8×8 二维矩阵。若给定 A , B , 计算 $C=AB$ 的 64 个分量 C_{ij} , $0 \leq i \leq 7$, $0 \leq j \leq 7$, 其代数式为

$$C_{ij} = \sum_{k=0}^7 a_{ik} \square b_{kj} \quad (0 \leq i, j \leq 7)$$

若在 SISD 计算机上要用 k , i , j 三重循环才能完成, 每重循环执行 8 次, 共需用 512 次加、乘时间。在阵列机上执行时, 利用 8 个 PE 部件并行计算, 则 j 循环一次即可完成, i , k 循环依旧, 因此只需 64 次加、乘时间, 速度提高 8 倍。程序流程图如图 5-53 所示。由于 8 个 PE 执行同一套指令, 故控制部件执行的 PE 类指令表面上是标量指令, 但实际上等效于向量指令。其次, A , B , C 向量各分量在处理单元存储器中的分布如图 5-54 所示。做乘法时, $B(k, j)$ 都从本处理单元的 PEM 中取出, 但被乘数 $A(i, k)$ 对所有处理单元是同样的。因此, 要利用阵列机的“播放”功能, 把当前 k 次循环中的 $A(i, k)$ 取出后送到 8 个 PE 的寄存器中。

(4) 累加和问题。将 n 个数的顺序相加过程变为并行相加过程。在 SISD 计算机上, 要进行 n 次相加, 而在并行处理机上, 采用递归相加算法, 则只需 $\log_2 n$ 次即可。如 $n=8$, 则并行处理机只需 $\log_2 8=3$ 次即可。设原始数据 $A(i)$ ($0 \leq i \leq 7$) 在 8 个 PEM 的 a 单元内, 它将按下列步骤运算:

第 1 步 置全部 PE 为“活动”态;

第 2 步 全部 $A(i)$ ($0 \leq i \leq 7$) 从 PEM 的 a 单元读入相应 PE 的寄存器 (RGA);

第 3 步 令 $k=0$;

第 4 步 全部 PE 的 RGA 内容送至传送寄存器 RGR;

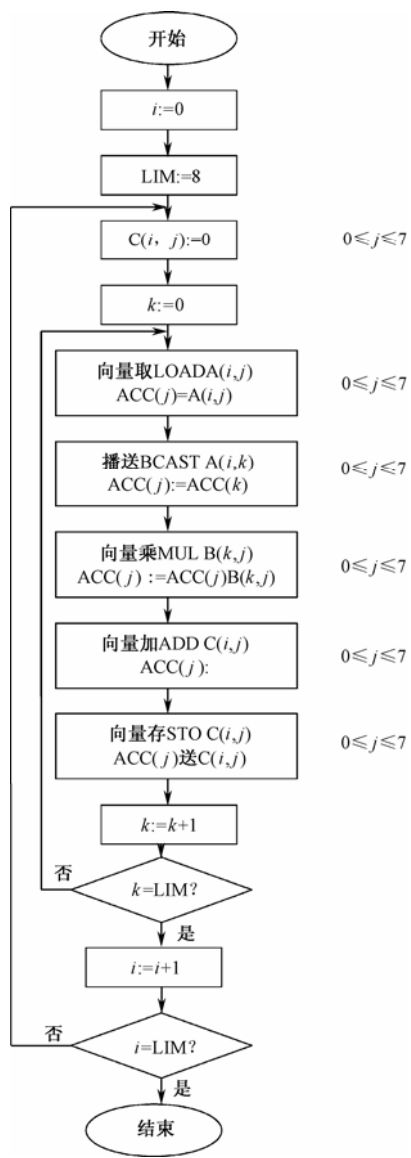


图 5-53 矩阵乘程序流程图

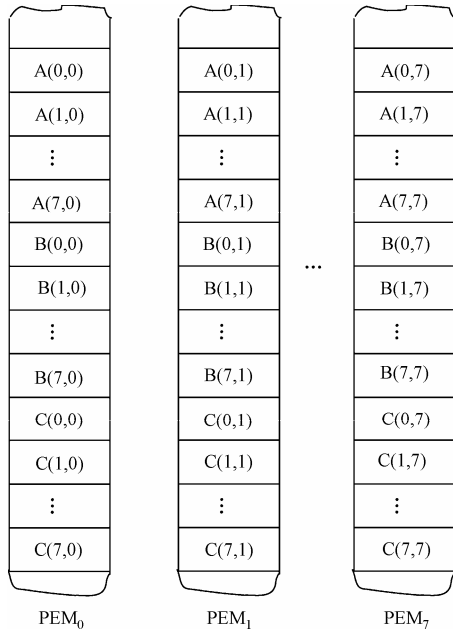


图 5-54 矩阵乘存储器分配

- 第 5 步 全部 PE 的 RGR 经过互连网络向右传送 2^k 步距；
 - 第 6 步 $j=2^k-1$ ；
 - 第 7 步 置 $PE_0 \sim PE_j$ 为“不活动”态；
 - 第 8 步 活动的 PE 执行 $(RGA) + (RGR) \rightarrow RGA$ ；
 - 第 9 步 $k+1 \rightarrow k$ ；
 - 第 10 步 如 $k < 3$ ，转第 4 步，否则下滑执行；
 - 第 11 步 置全部 PE 为“活动”态；
 - 第 12 步 全部 PE 的 RGA 内容存入相应 PEM 的 $a+1$ 单元中。
- 表 5-9 为各步计算结果。A(0)~A(7)在表中用 0~7 代表，第 5 步中 PE 的 RGR 内容右移超出 PE_7 就不表示了。此外，在快速傅里叶变换（FFT）、卷积、相关等处理大规模数组问题

方面，阵列机也能实现高效率的处理。

表 5-9 累加和各步计算结果

循环	k=0				k=1			k=2		
步骤	第 2 步	第 5 步	第 7 步	第 8 步	第 5 步	第 7 步	第 8 步	第 5 步	第 7 步	第 8 步
寄存器 PE	RGA	RGR	活动位	RGA	RGR	活动位	RGA	RGR	活动位	RGA
0	0		0	0		0	0		0	0
1	1	0	1	1+0		0	1+0		0	1+0
2	2	1	1	2+1	0	1	2+1+0		0	2+1+0
3	3	2	1	3+2	1+0	1	3+2+1+0		0	3+2+1+0
4	4	3	1	4+3	2+1	1	4+3+2+1	0	1	4+3+2+1+0
5	5	4	1	5+4	3+2	1	5+4+3+2	1+0	1	5+4+3+2+1+0
6	6	5	1	6+5	4+3	1	6+5+4+3	2+1+0	1	6+5+4+3+2+1+0
7	7	6	1	7+6	5+4	1	7+6+5+4	3+2+1+0	1	7+6+5+4+3+2+1+0

5.4.3 阵列处理机举例

阵列机在发展过程中出现了相当多的代表机型，前述的 ILLIAC IV 是其中具有典型意义的机器。表 5-10 列举了三种典型的 SIMD 计算机，从系统结构和性能，语言、编译器和软件支持等方面进行了比较。这些系统的 PE 数从 DAP 610 的 4096 到 MP-1 的 16384 及 CM-2 的 65536，阵列不断扩大，而 CM-2 和 DAP 都采用位片阵列，实现了细粒度处理。

表 5-10 三种典型的 SIMD 计算机比较

系统型号	SIMD 计算机结构和性能	语言、编译器和软件支持
MasPar 公司 MP-1 系列	可用的配置为 1024 ~ 16384 台处理机，达 26000MIPS 或 1.3GFLOPS。每个 PE 是 RISC 处理机，带 16KB 本地存储器。X-Net 网络加一个多级交叉开关互连网络	FORTRAN77、MasPar FORTRAN (MPF) 和 MasPar 并行应用语言；X-窗口，UNIX/OS, 符号调试程序，可视化和动画片制作器
Thinking Machines 公司 CM-2	多至 65536 个 PE 位片阵列排成十维超立方体，每个定点为 4×4 网格，每个 PE 最多有 1M 位存储器，32 个 PE 模块之间共享 FPU 选件，峰值速度达 28GFLOPS，持续速度达 5.6GFLOPS	由 VAX,Sun 或 Symbolics 360 主机驱动，PARIS 支持的 Lisp 编译器、FORTRAN90、C ^o 和 [*] Lisp
Active Memory Technology 公司 DAP 600 系列	1K 位 PE 方形网格互连成 4096 个 PE 的细粒、位片 SIMD 阵列，正交 4-邻位链接，具有 20GIPS 和 560MFLOPS 峰值性能	由主机 VAX/VMS 或 UNIX FORTRAN-Plus 或 DAP 上 APAL 提供，主机的 FORTRAN77 或 C。与 FORTRAN90 标准有关的 FORTRAN-Plus

1. BSP计算机

BSP 计算机是美国 Burroughs 公司和 Illianos 大学于 1979 年研制的，是共享存储器结构的 SIMD 计算机的典型代表。BSP 是一台附属于系统管理机的后台机，其系统结构如图 5-55 所示。

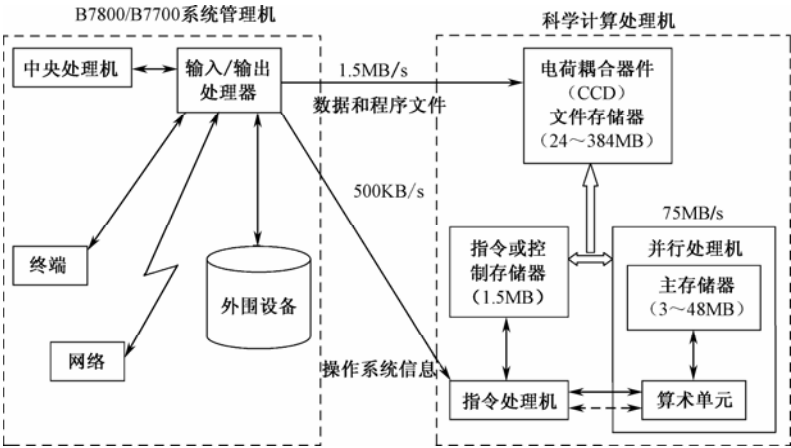


图 5-55 附属于系统管理机的 BSP 计算机

系统管理机在前台提供数据和程序编辑、与远程终端和网络的通信、BSP 程序的向量化编译和连接、数据存储数据库管理及分时服务等，而 BSP 在后台担任算术运算。BSP 由并行处理机、控制处理机、文件存储器、对准网络及并行存储器等组成，如图 5-56 所示。

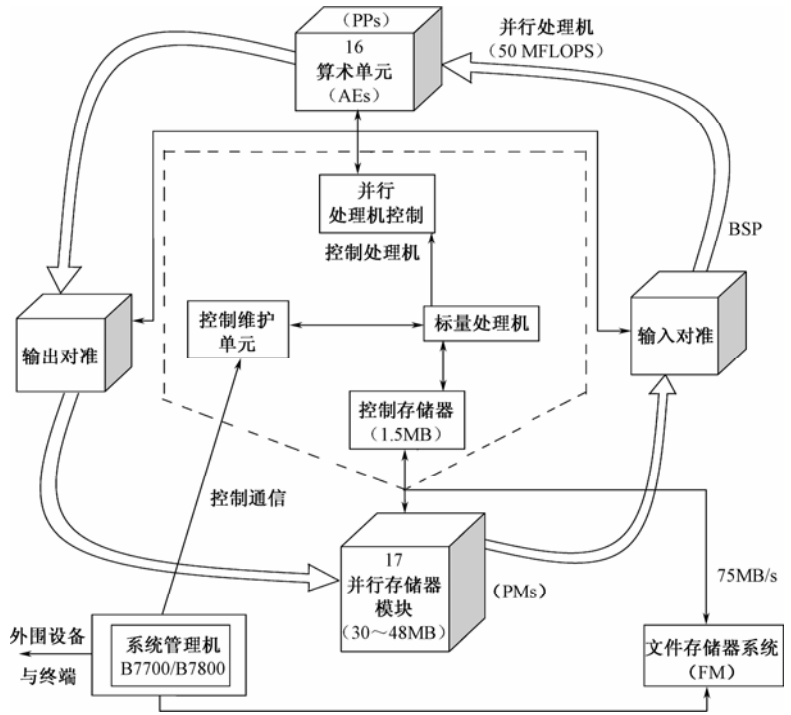


图 5-56 BSP 的功能结构与流水线处理

(1) 并行处理机。并行处理机有 16 个算术单元 AE，可对不同数组进行同一种指令操作。大部分算术运算在 2 个时钟周期内完成（每个时钟周期 160ns）。BSP 运算速度可达 50MFLOPS。数据存放在 17 个并行存储器模块中，每个模块容量为 512K×48，存取周期为 160ns，与 AE 之间以 100M×48/s（即 600MB/s）速率进行传输。17 个存储器模块构成一个无

冲突访问存储器, 允许对任意长度以及跳距不是 17 倍数的向量实现无冲突存取。16 个 AE 是以 SIMD 方式在单一微序列控制下同步工作。AE 使浮点加、减、乘法运算在两个时钟周期内完成, 使存储器频宽与 AE 进行三元操作 (三个操作数产生一个结果) 时的频宽相匹配。浮点除法运算要用 1200ns, 它是用 Newton-Raphson 迭代算法产生倒数来实现的。浮点数字长为 48 位, 尾数 36 位, 阶码 10 位, 尾符 1 位, 阶符 1 位, 以 2 为底, 数的精度可达十进制数的 11 位。AE 有双字长累加器和寄存器, 用硬件实现双精度运算, 还可以用软件实现三倍精度的运算。BSP 在用 FORTRAN 语言编制的计算程序中, 运算速度可达 20~40MFLOPS。除实现浮点操作以外, AE 还有较强的非数值处理能力。

(2) 控制处理机。用以控制并行处理机, 提供与系统管理机相连接口。标量处理机执行操作系统和用户程序的串行或标量部分, 时钟频率 12MHz, 运算速度可达 1.5MFLOPS。并行处理机的控制器对全部向量指令和某些成组标量指令经过合格检查后, 转换为微序列, 去控制 16 个 AE 操作。控制存储器容量为 256K×48, 存取周期 160ns, 每个字长 48 位, 另外加上 8 位奇偶校验位, 有单错纠正、双错检测 (SECDED) 能力。控制维护单元是系统管理机与控制处理机其余部分之间的接口, 用于初始化、监控命令通信和维护。

(3) 文件存储器。文件存储器是 BSP 直接控制下唯一的外围设备。BSP 的计算文件从系统管理加载到它上面, 然后进行排队, 由控制处理机加以执行。程序执行过程中所产生的暂存文件和输出文件在系统管理机输出之前, 存放在文件存储器中。文件存储器的数据传输速率较大, 缓解了 I/O 受限问题。

(4) 对准网络。由交叉开关、广播逻辑 (一个源广播到几个目的地) 和分解冲突逻辑 (几个源寻找一个目的地) 等硬件组成。存储器模块和对准网络的组合功能提供并行存储器无冲突访问。BSP 的浮点运算是流水进行的, 流水线由 5 个功能段组成。16 个操作数从存储器模块取出, 经过输入对准网络送给 AE 进行处理, 结果经过输出对准网络送回存储器模块, 这几级的操作都是重叠进行的。在物理上, 输入对准和输出对准都是在一个实际对准网络中进行。AE 也利用输出对准网络实现数据压缩、扩展以及快速傅里叶变换 (FFT) 算法等专用功能。除了 16 个 AE 表现出空间并行性以及流水处理的时间并行性外, AE 中向量运算还可以与标量处理机中标量运算并行, 使系统既适合长向量和短向量处理, 也能处理单独的标量, 因此系统功能很强也很灵活。

(5) 并行存储器。BSP 并行存储器由 17 个存储器模块组成, 存取周期为 160ns。由于每个周期可以同时存取 16 个字, 所以每个字的存取时间为 10ns。算术单元完成浮点加和乘运算需要两个变量, 并访问并行存储器两次, 因此其运算速度为 320ns/16 次, 即 20ns/次。算术单元中设有中间寄存器, 所以算术单元的运算速度与并行存储器的频宽能很好地平衡。由于程序和标量都存放在控制存储器中, 只有数组存取和 I/O 才用到并行存储器。对于三元向量来说, 因为两次算术运算中需要用到三个变量, 产生一个结果, 共访存 4 次, 所以在并行存储器与浮点运算之间的频宽也可保持平衡。对于长向量来说, 由于中间结果存放在寄存器中, 每次运算只需要一个操作数, 因此并行存储器有足够频宽留给 I/O 信息用。

BSP 采用线性向量法开拓并行性, 线性向量是反映其并行性的基本参量, 它以线性形式将元素映射到主存储器中。线性向量各分量的存储间距为常数 d 。例如, FORTRAN 按列映射时, 列分量的间距 $d=1$, 行分量的间距 $d=n$ (n 为列数), 正向对角线分量的间距为 $d=n+1$ 。BSP 在处理线性向量时既利用了空间并行性, 也利用了时间并行性。BSP 存储系统可以做到

在每个存取周期内给每个 PE 送一个有用的操作数，保证无冲突访问的硬件技术有：质数个存储器端口；存储器端口和 AE 间的完全交叉开关，以及按具体模式计算出有效地址的存储器地址生成机构等。地址模式是指正统的串行计算机所有的模式，即每个较高的存储地址是指存储器中的“下一个”字。设某个 BSP 机器有 N 个 AE 和 M 个存储器模块，存储器模块号为 μ ，线性地址为 a ， $\mu = a \pmod{M}$ ，在指定的存储器模块内的地址偏移量 $i = \lfloor a/N \rfloor$ 。现在设定某个类似 BSP 的机器 $N=6$ ， $M=7$ （在 BSP 中 $N=16$ ， $M=17$ ），有一个 4×5 矩阵以线性化向量映射到存储器中，其模块号 μ 和偏移量 i 的计算结果和存储映射如图 5-57 所示。模块号以一定范围（等于 N ，即 AE 数）重复出现，增量为 1。偏移量与同一模块重复出现的次数相对应，它在一个周期（等于 M ，即存储器模块数）内是不会重复的。例如图 5-57 中，数组元素 a_{11} 在 0 号模块（ $\mu=0$ ），偏移量为 0（ $i=0$ ），而 a_{42} 也在 0 号模块，偏移量为 1， a_{34} 也在 0 号模块，偏移量为 2。只要 AE 数（即 N ） \leq 存储器模块数（即 M ），则求得的偏移量值序列总可使不同的存储器模块与每一个 AE 连接。这时，每个 AE 存/取的数据都是唯一的。本例中 4×5 数组产生的存储模式如图 5-57 所示。现将数组第三行元素（ $a_{31} \sim a_{35}$ ）所对应的存储器模块号和偏移量计算如下：起始地址为 2，跳距为 $d=4$ ，因此可求得下列模块号：

$$\begin{aligned} \mu &= 2 \pmod{7}, 6 \pmod{7}, 10 \pmod{7}, 14 \pmod{7}, 18 \pmod{7} \\ &= 2, \quad 6, \quad 3, \quad 0, \quad 4 \end{aligned}$$

相应地，每个模块内的偏移量为

$$\begin{aligned} i &= \left\lfloor \frac{2}{6} \right\rfloor, \left\lfloor \frac{6}{6} \right\rfloor, \left\lfloor \frac{10}{6} \right\rfloor, \left\lfloor \frac{14}{6} \right\rfloor, \left\lfloor \frac{18}{6} \right\rfloor \\ &= 0, \quad 1, \quad 1, \quad 2, \quad 3 \end{aligned}$$

图 5-57 操作步骤如下：

- (1) 列出 4×5 矩阵 A ，如图 5-57 (a) 所示。
- (2) 将数组按列顺序排列，并给予顺序线性地址 a 。
- (3) 起始地址为每行第一个元素的线性地址 a ，跳距 $d=4$ ，按 $\text{mod } 7$ 求出该行的每个元素的模块号 μ 。
- (4) 用 $i = \lfloor a/N \rfloor$ 求出该行的每个元素的模块内偏移量 i 。
- (5) 将矩阵的每行数组元素的模块号 μ 和偏移量 i 求出，并画出 4×5 矩阵的物理映射表格，如图 5-57 (b) 所示。该表格上方为 μ （因为 $M=7$ ，所以是 $0 \sim 6$ ）表格左边为 i （模块内偏移量为 $0 \sim 5$ ），格子内从左到右为线性地址 a ，反向对角线格子为冗余，所以 a 从左到右填入，遇到冗余，则转到下一行。
- (6) 将每个数组元素按 a ， μ ， i 填入物理映射表格，如图 5-57 (c) 所示。

从图 5-57 的物理映射可看出，6 个 AE（ $N=6$ ）每次从 7 个存储器模块中的 6 个（ $M=7$ ）各取出一个数组元素而不发生冲突。

在并行递归算法中对向量子集各元素逐次按 2 的整数幂相间访问，如开始相联访问，然后按 2，4，8 等变址。对于这类算法，并行存储器的模块数取成 2 的整数幂就会在 2^i 变址时发生访存冲突。所以并行存储器的模块数应取成质数，才能较好地避免访存冲突。只要变址跳距与模块数 M 互质，访存就能无冲突地进行，BSP 并行存储器采用 $M=17$ ，所以是质数存储系统。由于每个存取周期只有一个存储器模块没有被访问，因此存储器频带的冗余度很小（ $N=16$ ， $M=17$ ，冗余度为 $1/17$ ）。对于一维数组的任何算术数列指标模式，除每一次排序第 17 个元素以外，对它可以无冲突访问。对于二维数组，若将错开距离（跳距）定为 4，则对

行、列、正向对角线、反向对角线以及其他普通划分都能无冲突访问。这一方法可推广到高维数数组。当数组元素的地址相隔是存储器模块数的整数倍时，则冲突一定会发生，因为所要访问的数组元素都处于同一个存储模块中。对 BSP 来说，应避免跳距为 17, 34, 51 等情况。如果在 BSP 中发生冲突，其运算仍可正确进行，但速度下降到正常速度的 1/16，而系统能记录冲突以及对总的运行时间的影响，提供给用户以便对程序采取改进措施。

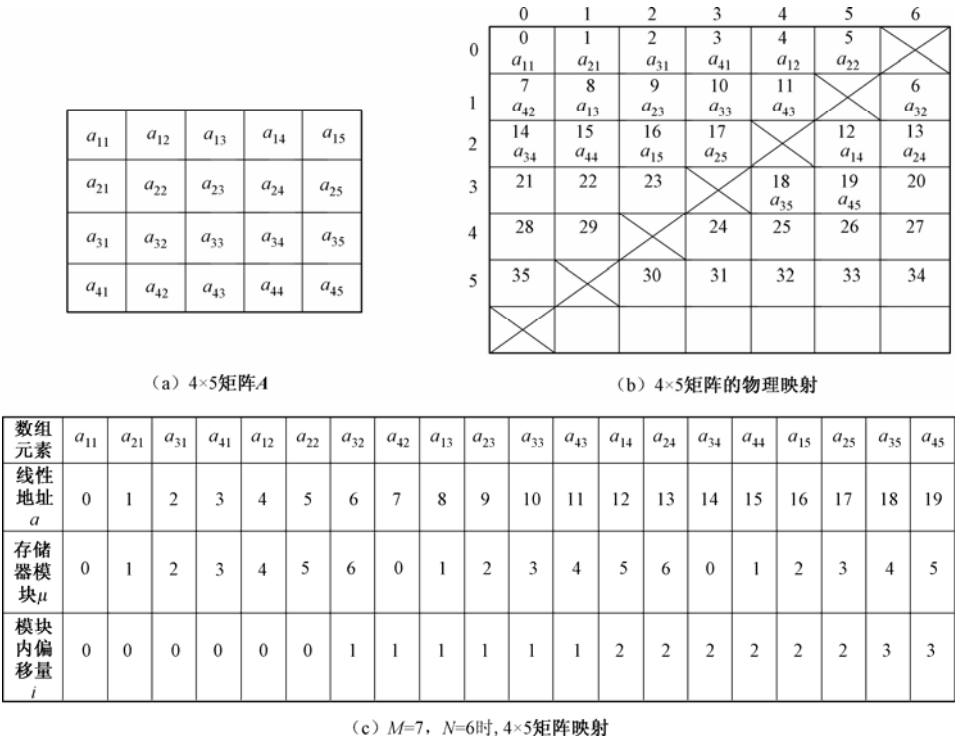


图 5-57 矩阵元素线性化向量的 BSP 存储器映射

- BSP 可以对下列 4 类操作实现并行计算：
- (1) 16 个算术单元 AE 实现并行计算。
 - (2) 存储器的存取以及存储器和算术单元间的数据传输可以并行进行。
 - (3) 在并行处理机控制器内的变址值计算，向量长度计算以及循环控制计算实现并行。
 - (4) 线性向量操作描述字在标量处理机中的生成。

2. CM-2 计算机

Connection Machine 公司的 CM-2 是一台细粒度的 SIMD 计算机，它由 65536 个位片 PE 组成，峰值速度达到 28GFLOPS，持续速度为 5.6GFLOPS。CM-2 系统结构如图 5-58 所示。其工作流程如下：前端机执行程序，当需要并行处理时，发送微指令到后端处理阵列，定序器 (Sequencer) 分解微指令并且把它们广播到阵列中所有数据处理器 (Data Processor)，处理结果回送定序器，由定序器将结果送前端机。前端机和处理阵列之间有三条总线：

- (1) 广播总线 (Broadcasting)，把数据或指令同时传送到所有数据处理器。
- (2) 全局组合总线 (Global Combining)，前端机通过该总线对来自各数据处理器的数据进行求和、求最大值、逻辑或等运算。

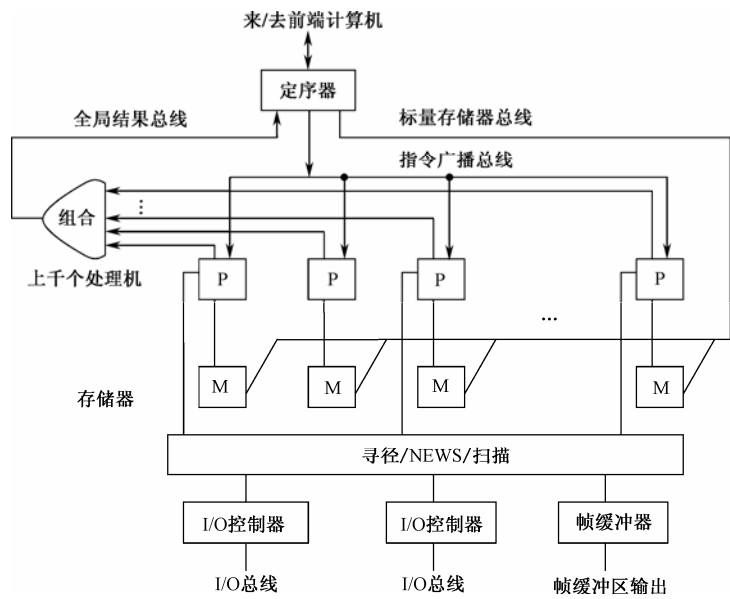


图 5-58 CM-2 系统结构

(3) 标量存储器总线 (Scalar Memory Bus)，前端机通过该总线与存储器进行数据通信，每次存取 32 位数据。

CM-2 前端主机可以是 VAX 机或 Symbolics 机，而后端机的处理阵列由 4096~65536 个刀片式处理器组成，所有处理器 (PE) 都由定序器控制。定序器对来自前端机的微指令进行译码，然后把毫微指令广播到阵列中各个 PE，所有 PE 可以同时访问本地存储器 M，PE 以锁步方式执行广播来的指令。PE 之间通过寻径、NEWS 网格 (NEWS Grid) 或扫描机构 (Scanning Mechanism) 相互交换数据，这些网络也与 I/O 接口相连。大容量外存称为数据穹 (Data Vault)，由磁盘组构成，容量达 64GB。CM-2 处理器结点如图 5-59 所示。每个结点有 32 个 PE、一个可选浮点加法器、存储器芯片和 PE 间通信接口。PE 用有 3 个输入和 2 个输出的位片 ALU、锁存器和存储器接口实现。ALU 可以执行位串全加和布尔逻辑操作。每个结点有两片处理器芯片，共享一组存储器芯片，每个处理器芯片有 16 个 PE，数据通路为 22 位 (16 位数据和 6 位 ECC)，存储器地址 18 位，允许 32 个 PE 共享 256K×16 (512KB)，浮点芯片一次执行 32 位的操作，中间结果可存入存储器供后续使用。每个处理器芯片有一个寻径器，用于 PE 间数据寻径。在 CM-2 最大配置时，所有处理器芯片上共有 4096 个寻径器结点，可以连成一个 12 维超立方体。每个处理器芯片中的 16 个物理 PE 可以排列成 8×2，1×16，4×4，4×2×2，2×2×2×2 等形式的网格。NEWS 网格建立在每个 PE 都有上、下、左、右 4 个邻居的基础上，把每个结点的内部网格与全局网格结合在一起就可以把 PE 排列成任意维、任何形式的 NEWS 网格，使数据非常有效地沿着应用要求而建立的专门网格快速传递。除了通过超立方体寻径器动态重构 NEWS 网格外，CM-2 还有专用硬件扫描机构支持对整个 NEWS 网络的扫描或传播，扫描是把通信和计算结合在一起。可以沿某一维方向对网格的每一行扫描，求出该行部分和，找出最大值或最小值，进行与、或、异或计算。扫描操作可以扩展到对整个阵列的所有元素。传播能将一个数据传送给其他芯片上的 PE。一位二进制数只用 75 步就可以沿着超立方体连线从一个芯片送到所有其他芯片。CM-2 有 2~16 条高速 I/O

通道用于数据、图像 I/O 操作。外设包括数据穹、CM-NIPPI 系统、CM-IOP 系统和 VME 总线接口控制器。CM-2 用于大规模并行处理 (MPP) 所面临的应用课题包括：借助相关反馈技术的文档检索、基于记忆的推理、医疗诊断系统 QUACK、自然语言处理、SPICE 的 VLSI 电路分析和布线、流体力学计算、信号/图像/视觉处理和集成、神经网络模拟和连接模型、动态规划、上下文无关文法分析、射线追踪图以及计算几何等。

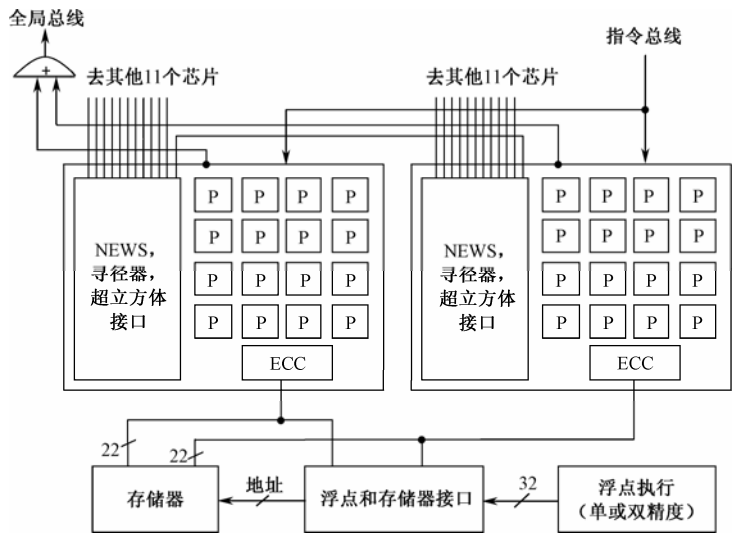


图 5-59 由两个处理器芯片、一组存储器和浮点芯片组成的 CM-2 处理器结点

3. 大规模并行处理机

1979 年，美国国家航天局 Goddard 中心与 Goodyear 航天公司合作研制一台用于处理遥感卫星图片的大规模 SIMD 阵列机获得成功。由于这台机器用了 $128 \times 128 = 16384$ 个可并行工作的微处理机，因此被定名为大规模并行处理机 MPP (Massively Parallel Processor)。MPP 可对变长的操作数按位片进行算术运算。MPP 有一个微程序控制器，能够十分灵活地定义向量、标量和 I/O 操作的指令系统，整个 MPP 系统均用微处理器芯片和 SRAM 芯片组成。

MPP 系统结构如图 5-60 所示。阵列部件 ARU (ARray Unit) 由 128×128 个 PE 构成一个二维阵列，以 SIMD 方式工作。每个 PE 有一个 1027 位 SRAM，有奇偶校验功能。每个 PE 是位片式微处理机，与四周近邻相连。程序员可在平面、水平圆柱、垂直圆柱、开螺线、闭螺线等 5 种阵列拓扑中任选一种，增加了阵列机结构的灵活性。在阵列中增加了 4 列冗余 PE，使阵列的物理结构为 132 列 \times 128 行。当阵列硬件出现故障时，可用旁路掉故障列的方法，使阵列逻辑结构仍为 128×128 。每个 PE 内有一个串行加法器，并用一个移位寄存器实现位串式加法。PE 阵列的时钟周期为 100ns。阵列控制器 ACU 是微程序控制器，对 PE 阵列处理进行管理，完成标量运算以及控制数据在 PE 阵列上的移位。程序和数据管理部件 PDMU (Program and Data Management Unit) 是一台后端小型计算机，其作用是管理阵列中的数据流，将程序装入控制器，进行系统的测试和诊断并提供程序开发手段等。

MPP 系统运行方式有两种：① 独立方式在终端予以操作控制；② 在线方式由外接计算机予以控制。MPP 与外接计算机之间的数据传输速率为 6MB/s，按高速数据方式运行时，数据通过 128 位外部接口传输，其速率可达 320MB/s。

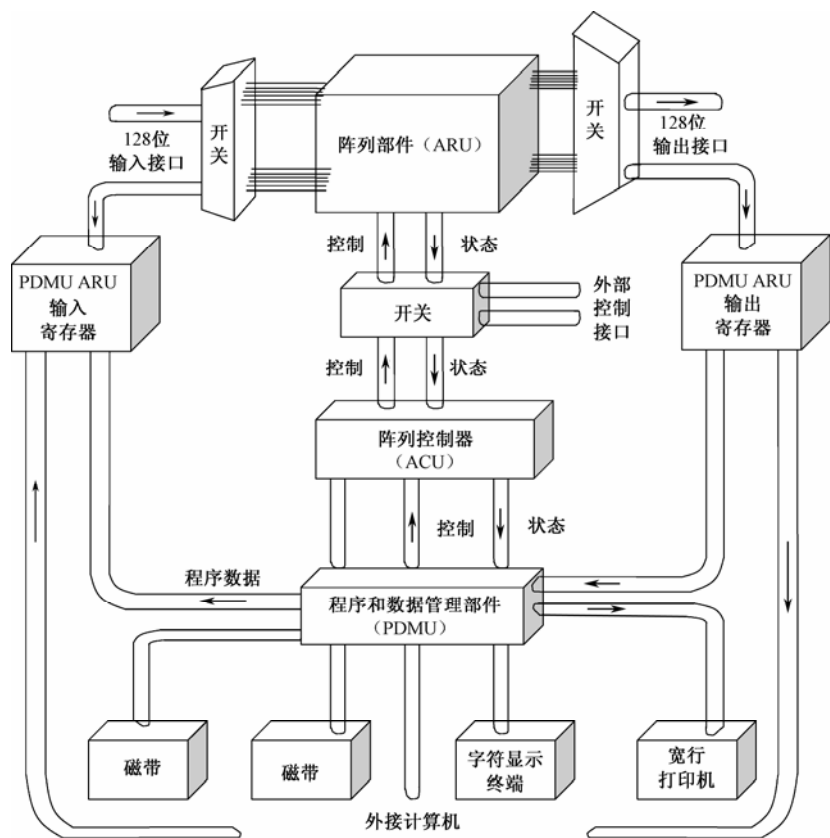


图 5-60 MPP 系统结构

4. MP-1 系统

MasPar 公司的 MP-1 系列是一台中粒度 SIMD 计算机，它的系统结构如图 5-61（a）所示。MP-1 系统结构由 4 个子系统组成：PE 阵列、阵列控制部件（ACU）、具有标准 I/O 的 UNIX 子系统和高速 I/O 子系统。UNIX 系统负责传统的串行处理。PE 阵列和高速 I/O 配合负责大规模并行计算。MP-1 系列有 1024，4096 个乃至最多可达 16384 个 PE 的各种配置。当配置为 16K 个 PE 时，32 位 RISC 整数操作时峰值速度是 26000MIPS，单精度浮点运算速度为 1.5GFLOPS，双精度浮点运算速度为 650MFLOPS。

（1）PE 阵列。一个 PE 群（PEC）有 16 个 PE，64 个 PEC 有 1024 个 PE，和有关存储器一起构成处理器板，其连接形式如图 5-61（b）所示。每个 PEC 芯片通过 X-Net 网格型网络和全局多级交叉开关寻径器网络（即 S_1 ， S_2 和 S_3 ）与 8 个相邻的 PEC 芯片相连。PEC 内部结构框图如图 5-62（a）所示。PEC 内有 16 个 PE 和 16 个局部存储器 PMEM。16 个 PE 排列成 4×4 二维网格型互连阵列。群内 16 个 PE 共享多级交叉开关寻径器的一个访问端口。PE 间通信有三种机制：第一种机制是 ACU-PE 阵列通信，ACU 把指令/数据同时广播到阵列中所有 PE，并且对并行数据进行全局归约，从阵列中收回标量值；第二种机制是 X-Net 近邻通信，X-Net 将每个 PE 与它的二维网格上的 8 个相邻 PE 直接相连，X-Net 的总通信频宽为 18GB/s；第三种机制是全局交叉开关寻径器通信，每块 PE 板有 S_1 ， S_2 和 S_3 三个寻径器芯片，三级寻径器实现一个 1024×1024 交叉开关功能，源 PE 经过 S_1 ， S_2 和 S_3 到达目的 PE，当 MP-1

全配置时有 1024 个 PEC，每级有 1024 个寻径器端口，每个寻径器同时支持 1024 个连接，能实现所有 PE 之间的全局通信，总通信频宽为 1.3GB/s。

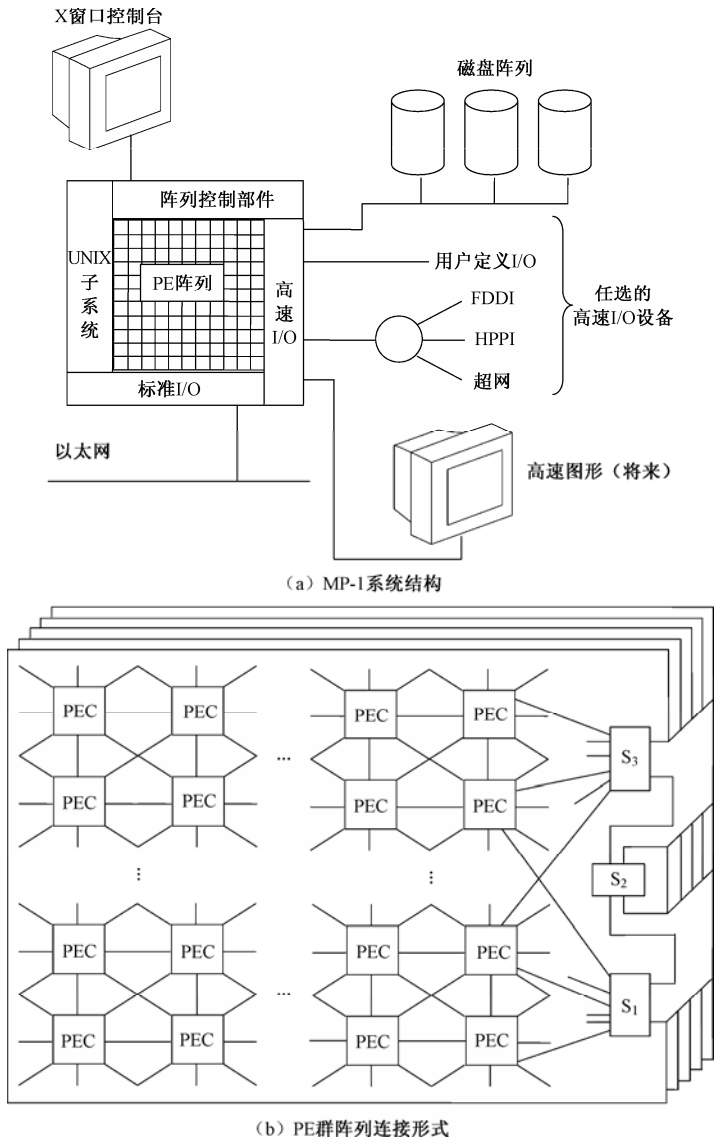


图 5-61 MP-1 系统结构

(2) 处理单元和存储器。PE 的结构如图 5-62 (b) 所示。PE 中设有取指和译码逻辑，整数和浮点数计算在每个基于 RISC 结构的 PE 中执行。存取数指令将数据在 PEM 和寄存器组之间传送。每个 PE 有 40 个供程序员使用的 32 位寄存器和 8 个供系统使用的 32 位寄存器。寄存器可按位或按字节寻址。PE 有 4 位整数 ALU、1 位逻辑部件、64 位尾数部件、16 位指数部件（即阶码）和标志部件（即 FLAGS）。位总线宽度为 1 位，NIBBLE 总线宽度为 4 位。PEM 有直接或间接寻址，最大频宽为 12GB/s。每个 PE 的大多数数据在位总线和 NIBBLE 总线上传送，每个微操作可同时激活 PE 内不同功能部件，因此，PE 可同时执行整数、逻辑运算和浮点数操

作。PE 本身运行的时钟频率不高，与 CM-2 类似，靠大规模并行处理提高系统速度。

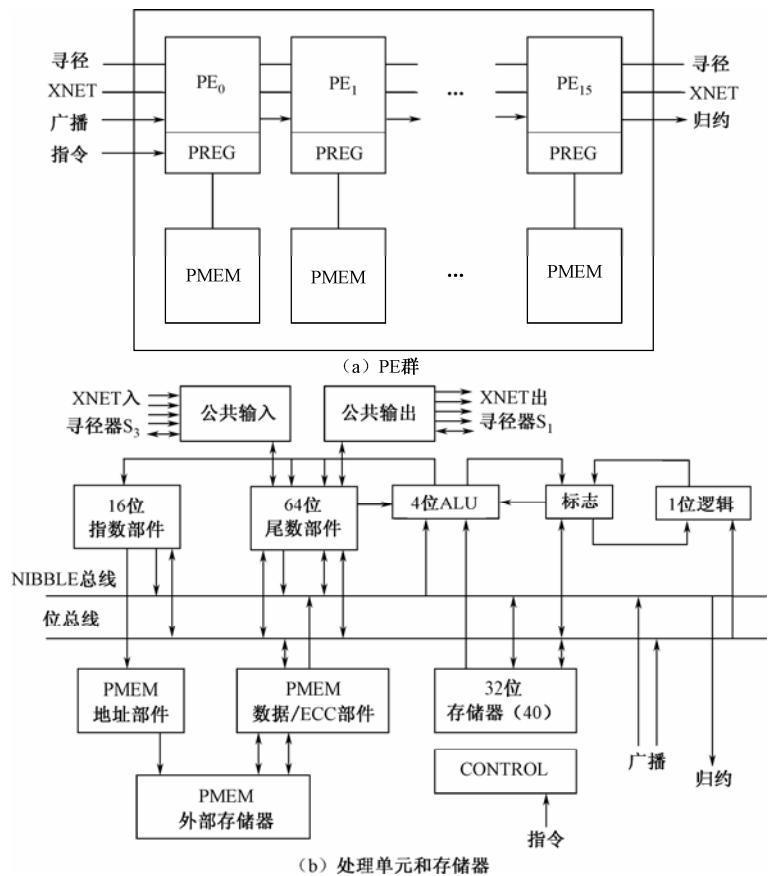


图 5-62 MP-1 处理单元和存储器的设计

（3）阵列控制部件。ACU 是一台标量 RISC 处理机，速度为 14MIPS。ACU 负责取指、指令译码、计算有效地址、处理标量数据、发送 PE 阵列的控制信号及控制 PE 阵列的状态，因此 ACU 与 CM-2 中的定序器类似，ACU 采用微程序控制，有一个称为存储器机器的功能部件与 ACU 并行工作，它执行 PE 阵列的装入和存储操作，而 ACU 将算术逻辑运算指令和寻径指令广播到各个 PE 并行执行。ACU 标量指令执行时间为 70ns。

（4）并行磁盘阵列。PE 阵列通过高速 I/O 子系统与并行磁盘阵列通信。高速 I/O 子系统用全局寻径网络实现，其速度达到 1.3GB/s。磁盘阵列格式化容量为 17.3GB，速度为 9MB/s。磁盘阵列支持数据并行计算，提供透明文件系统以及多级容错能力。

近年来，世界各国纷纷投入大量人力、物力研制高性能计算机，成果日新月异。2008 年 7 月，美国 IBM 公司和美国能源部宣布已研制出新一代全球最快计算机——“走鹃”。该机由 IBM 公司和美国洛斯阿拉莫斯国家实验室共同研发，共采用 116640 个处理器，其中 12960 个为经过改良的 IBM Cell 8 核 MPU（微处理器）。“走鹃”占地 557m²，重 226.8 吨，连接光纤长 91.7km，存储空间 80TB（80 万亿字节），功耗近 3MW。“走鹃”是世界上第一台突破“Petaflop”障碍的计算机，其峰值达到 1000TFLOPS（即运算速度 1150 万亿次/秒，实测峰值为 1456 万亿次/秒）。我国于 2008 年 7 月宣布中科院计算所曙光公司研制成功“曙光 5000”，

采用 6600 个 4 核 AMD MPU，峰值达到 230TFLOPS（230 万亿次/秒），占地 75m²，共 75 个机柜，每个机柜内集成 200 个 CPU 芯片，单机柜计算能力达到 7.5TFLOPS，（而“走鹃”不到 4TFLOPS），整机功耗 0.7MW，平均每百万亿次的能耗比“走鹃”低 10% 左右。所以，“曙光 5000”在“系统密度”和能耗指标上超过了“走鹃”。

5.5 相联处理机

相联处理机是基于存储器操作并行的 SIMD 并行处理机。第 3 章介绍和分析了并行主存系统（3.3 节），多体并行存储器是存储器并行操作的一种形式，采用多体并行存储器的目的是提高 CPU 发送指令流和数据流的速度，而如何处理指令和数据与存储器本身的结构无关。是否采用多体并行存储器不是区分单处理机系统和多处理机系统的根本依据。但是，如果存储器内部具有信息处理功能，情况就不同了。当然，存储器处理信息存在着串行进行还是并行进行的问题。按内容访问的相联存储器，就是带信息处理的存储器。它按信息内容的部分特征或全部特征，在存储器内进行比较、符合、分解等处理，将内容与特征相符的所有存储单元在一次访存中都找出来。在这一过程中必须依赖并行处理技术。本节将介绍相联存储器和相联处理机组成、结构及特点，相联检索算法以及典型相联处理机系统。

5.5.1 相联处理机结构

1. 相联处理机的组成和特点

以相联存储器为核心，配上必要的中央处理部件、指令存储器、控制部件和 I/O 接口，就构成了一台以存储器并行为特征的相联处理机（Associative Processor）。相联处理机构成如图 5-63 所示。相联处理机除完成信息检索任务外，还可以解一系列数学计算问题，其中中央处理部件也是并行操作的，计算效能比传统的顺序处理计算机要高。相联处理机有两个显著特点：

（1）相联处理机是按信息的全部或部分特征（即关键字）并行地对存储器内的许多单元（字）进行访问，与各单元的有关字段内容进行比较，根据两者是否一致（符合）即可找到所需要的全部数据或数组。由于这些操作对存储器各个单元是并行进行的，单元的地址对访存并不重要，而存储的内容却成了访存的依据，与传统的按地址顺序访存完全不同。

（2）控制部件命令能对许多数据同时进行算术或逻辑运算，这就要求相联存储器单元不仅有存储功能，还应有信息处理能力，即每个单元必须有一个处理功能部件，才能实现并行操作，尽管它的处理速度比传统的 CPU 速度要低，但由于它是重复设置且并行工作，因此整体速度却可以很高。由于相联处理机能对多个数据同时执行一条指令的运算，因此能替代传统顺序计算机编制的程序中的循环程序段，简化了程序，减少出错，提高了程序的可靠性。

相联处理机的缺点是硬件成本高，特别是相联存储器容量增大时，成本显著增加，所以相联处理机的规模受到限制，往往和传统的串行工作的计算机系统（前台机）配合在一起使

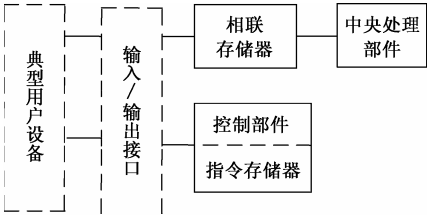


图 5-63 相联处理机的构成

用, 作为后台机, 专门处理需要相联检索的高速并行处理任务, 从而大大提高整个系统的处理能力和工作效率。

2. 相联存储器的组成

相联存储器组成如图 5-64 所示。它由下列部件组成:

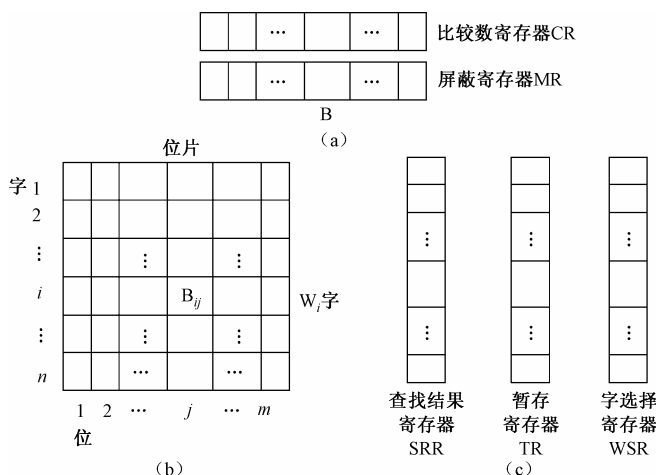


图 5-64 相联存储器的组成

(1) $n \times m$ 二维位存储阵列。由位片构成, 有 n 个字, 每个字有 m 位, 阵列中每个位单元 B_{ij} 是一个与比较逻辑门和读/写控制电路相连的触发器, 其典型逻辑电路如图 5-65 所示。 B_{ij} 可以清 0、写入、读出或与外部关键字相应位进行比较。

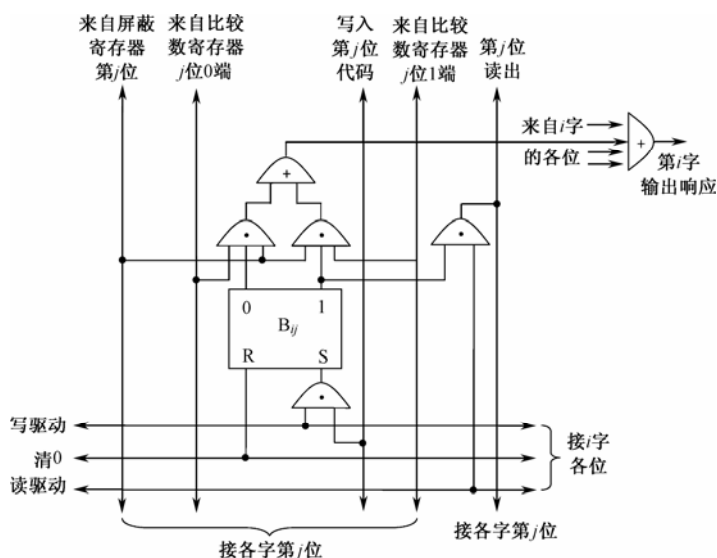


图 5-65 相联存储器位单元的逻辑电路

(2) 屏蔽寄存器 **MR**。设置屏蔽码, 控制相对应位不参加比较。当 **MR** 的第 j 位为 0 时, 就使存储器阵列的第 j 个位片不参加比较 (即阵列中所有字的第 j 位都不参加比较)。

(3) 比较数寄存器 CR。存放比较时用的关键字。

(4) 查找结果寄存器 **SRR**。它是一个 n 位的寄存器，用于寄存本次查找结果。当阵列中第 i 个字与关键字不匹配时，就将 **SRR** 的第 i 位清 0，否则，置 1。

(5) 暂存寄存器 **TR**。有一个或多个，保存上一次或前 n 次查找结果，以便处理机对若干次查找结果做进一步的“与”、“或”、“非”等逻辑运算。

(6) 字选择寄存器 **WSR**。控制阵列中哪些字不参加本次查找（比较）。若 **WSR** 的第 i 位为 1，则对应的第 i 个字参加查找；否则，不参加查找。

上述所有寄存器都可以置“1”、复位或从外部装入二进制信息。

存储器阵列的外围电路还有匹配检测电路和优先级逻辑电路，它们可在位片之间和查找结果的指示模式之间进行向量逻辑运算。相联检索步骤如下：

(1) 开始检索前，将关键字置入 **CR** 寄存器；

(2) 根据存储器位片参与比较的需要设置屏蔽寄存器 **MR** 和字选择寄存器 **WSR**；

(3) 发出查询信号，由相联存储器硬件完成有关位片上参与检索的所有存储器字与 **CR** 内关键字的并行比较；

(4) 根据比较结果，由硬件将查找结果寄存器 **SRR** 各位置成相应状态。

相联存储器可以分成两种构型：一种是“全并行”方式，即字向和位向的比较全部并行，其特点是操作简单、速度快，缺点是硬件非常复杂、成本高；另一种是“位片串字并”方式，即位片比较是顺序串行，字向比较仍为并行，其特点是硬件省、价格低，缺点是速度较慢，当处理的字数比每个字的位数多得多时，速度比全并行下降不多。

5.5.2 相联检索算法

相联存储器可以完成全等/不等、小于/大于、不小于/不大于、最小值/最大值、区间内/区间外、次小值/次大值、仅小于比较数/仅大于比较数、次小于比较数/次大于比较数等多种比较和查找操作，用相应的逻辑算法实现。若将查找结果或产生响应的单元取出，用另一类算术算法处理，就可以实现更为复杂的运算和处理。下面以“位片串字并”方式的相联存储器上实现全等查找、最大值查找、幅值比较查找等为例，介绍相联检索逻辑算法的基本思想。

1. 全等查找算法

所谓全等查找，是指找出与 **CR** 未屏蔽部分内容完全相同的全部字单元，其算法如下：

(1) 设置 **WSR**，控制只对 $WSR_i=1$ 的那些字进行操作；

(2) 设置 **MR**，控制只对 $MR_j=1$ 的那些位片段参与全等比较；

(3) 将比较查找关键字装入 **CR**；

(4) 对 $MR_j=1$ 的那些位片段逐位进行相联查找，当 $CR_j=1$ 而 $B_{ij}=0$ 或者 $CR_j=0$ 而 $B_{ij}=1$ 时，即比较结果不相等，查找产生的信号将 WSR_i 清 0；

(5) 查找完毕，**WSR** 中标志位仍为 1 的存储单元就是全等查找的响应单元。

2. 最大值查找算法

所谓最大值查找，就是找出存储器中所存的最大数及存放最大数的所有单元，其算法如下：

(1) 将 **WSR** 置成全 1，**CR** 置初始值为全 1；

(2) 将 **MR** 最高位置成 1，其余位置为 0；

(3) 然后进行比较，检查各待查单元的最高位是否为 1，若有响应，表明最大值的最高位为 1，同时，将所有未响应单元对应的 WSR_i 清 0，使其不参加下一个位片的查找；

(4) 若没有一个响应，表明最大值的最高位为 0，此时，将 CR 的该位清 0，WSR 中内容不变；

(5) 将 MR 的次高位置成 1，其余位置为 0，然后进行类似 (3)，(4) 的操作；

(6) 经过自左至右逐位比较处理，最后，CR 中保留的内容就是要找的最大值，WSR 中的状态就是存放此最大值所在存储单元的位置。

最小值查找与最大值查找类似，其不同点是 CR 初始值为全 0，从最高位开始逐位比较，若有响应，表明最小值该位为 0，若没有一个响应，将 CR 中该位置成 1。最后，CR 中就是要找的最小值，WSR 中的状态就是存放此最小值所在存储单元的位置。

3. 幅值比较查找算法

幅值比较查找是指给定比较数后，分别找出大于、等于、小于该比较数的单元。为此，可对每个单元设置一个三位的标志位，开始前，各单元的标志位为 100，表示“未定”状态。然后自左至右逐位查找，会出现下述三种情况：

(1) $CR_j=0, B_{ij}=1$ ，表示 i 单元第 j 位的值大于比较数第 j 位的值，即 i 单元的值大于比较数，将该单元的标志位置成 010，以后，该单元不再参加比较；

(2) $CR_j=1, B_{ij}=0$ ，表示 i 单元第 j 位的值小于比较数第 j 位的值，即 i 单元小于比较数，将该单元的标志位置成 001，以后，该单元不再参加比较；

(3) $CR_j=B_{ij}$ ，表示 i 单元第 j 位的值等于比较数第 j 位的值，该单元的标志位维持 100，并继续参与下一个位片的比较。

将所有单元从最高位到最低位全部查找一遍，根据各单元的标志位，就可以区分出哪些单元内容大于比较数，哪些单元内容小于比较数，哪些单元内容等于比较数，予以统计、分析。

4. 其他算法

有了上述基本查找算法，不难产生其他相近算法。

(1) 仅大于比较数查找算法。首先利用幅值比较查找算法找出大于比较数的全部单元；然后再利用最小值查找算法找出这些单元中的最小值，这些（个）具有最小值的单元必定是仅大于比较数的那些（个）单元。

(2) 仅小于比较数查找算法。类同于仅大于比较数查找算法。首先利用幅值比较查找算法找出小于比较数的全部单元；然后再利用最大值查找算法找出这些单元中的最大值，这些（个）具有最大值的单元必定是仅小于比较数的那些（个）单元。

(3) 排序算法。基本思想是利用最小值/最大值查找算法，然后进行交换，实现升序/降序排列。升序算法如下：先找出其中最小值所在单元，然后将其与第一个单元内容交换，再找余下单元中最小值单元，与第二个单元内容交换，……如此进行到 $n-1$ 个字完成后，第 n 个字必定是最大值，这样存储器内容就完成了按值由小到大的排序。降序算法类似于升序算法，其不同之处仅在于利用最大值查找算法，然后依次交换，最后完成按值由大到小的排序。

(4) 插入/删除算法。在已排好序的记录 A, B, C, D 中插入一个新记录 X，于 B, C 之间，得到 A, B, X, C, D，或者删除记录 C，并去除间隔，得到 A, B, D。在传统的按地址访问的机器上，使用指针链表结构，通过修改链表指针值来实现插入或删除，不仅会增加存放链表的存储空间，而且链表查找时间也较长。在相联处理机上可采用下列算法：先用全等查找算法找到存储记录 B 的单元；然后用大于等于 B 记录号的查找算法，将这些单元的记录号增 1；最后将需插入的记录 X 给予 B 的原记录号予以存入。删除算法只是插入算法的逆

过程：先用全等查找算法找到存储记录 C 的单元；然后用大于等于 C 记录号的查找算法，将这些单元的记录号减 1；最后释放原存储记录 C 的单元。不论链表有多长，插入和删除的步骤总是这几步。由此可见，在相联处理机上，按内容进行信息检索和更新的效率要比传统的按地址访问的机器高得多。

除了逻辑算法外，也有不少在相联处理机上运算的算术算法，如求反、加常数、字段相加、乘常数等，以及按内容进行高速检索和更新、矩阵运算、线性方程求解等较为复杂的算法。所以相联处理机在算术运算上有比传统机器更高的效率。

5.5.3 相联处理机举例

1. PEPE 系统

并行单元处理集合机 PEPE (Parallel Element Processing Ensemble) 是由 Bell 实验室、系统发展公司等单位为美国陆军弹道导弹防御高级技术中心设计、研制和生产的专用系统。该系统安装在“爱国者”拦截导弹系统内，对雷达搜索到大批量目标回波与计算存储的原有目标数据进行比较、识别、分类和精确跟踪，以便向拦截导弹发出制导命令。该系统也适用于天气预报、空中交通管制、图像数据处理及具有并行性而又有大量计算的场合。雷达数据处理实质上是对大型、复杂数据库进行实时更新，对相互独立的数组进行计算，这些计算对所有目标又都是类似的，因此具有并行处理的内在特性。再者，为了确定对雷达脉冲的要求，还必须对目标文件按优先权进行多维检索，以便按战斗环境的要求有效地调度雷达脉冲。采用相联存储器有很高的检索效率，因此 PEPE 系统采用全并行相联处理机体系结构。全并行相联处理机可以分为字结构和分布逻辑两种类型。PEPE 系统属于分布逻辑类型，是一种面向字符的全并行相联处理机。存储器中的比较逻辑与具有固定位数的字符单元或字符串单元联系在一起，因此减小了硬件复杂性，造价相对较低。

PEPE 系统逻辑结构框图如图 5-66 所示，它由输出数据控制、运算控制器、相关控制器、相联输出控制器、单元存储控制、控制系统及大量处理单元构成。PEPE 系统共有 8 个单元集合体，每个集合体有 36 个处理单元，所以系统共有 288 个处理单元。主机 CDC-7600 通过三个标准的 I/O 多路转接器通道与三个 PEPE 控制机架（即控制器）相连。主机通过操作系统对 PEPE 系统进行控制，将程序和原始数据送至 PEPE 存储器，分配和调度适当的任务。主机有执行顺序任务、编译和对外设 I/O 等功能。

控制机架中相关控制器控制各向处理单元 (PE) 输入数据，运算控制器用于控制各 PE 中运算部件操作，相联输出控制器用于控制各 PE 的数据输出，它们各自独立工作，因此控制机架可视为一个多处理器系统。每个控制机架有一个模块化的主机接口，更换接口部件就可连到不同主机上。控制机架将指令流分成两部分：顺序部分在机架内执行，并行部分则在并行单元集合体内执行。每个 PE 由相关、算术运算、相联输出三个部件和一个本地数据存储器组成。相关部件用于输入数据，它的指令特别适合将新数据与原文件中数据进行快速相联检索。算术运算部件的指令系统与一般通用机类似，有定点和浮点算术运算指令、存数和取数指令、逻辑操作指令等。相联输出部件用于数据排序和输出，并能对复杂的多维文件进行快速检索。这三个执行部件受控制机架中相应控制器控制，可以同时工作。数据存储器容量为 1024×32 ，受控制机架中单元存储控制部件的控制。288 个 PE 集合体中，各 PE 间没有通信功能。因为对弹道导弹防御系统来说，各目标之间没有通信的必要，这是与阵列机的区别之处。由于各 PE 独立工作，一个 PE 失效不

影响整个系统的运行，因此系统有较高可靠性。由于系统对顺序和并行的任务分别由独立的、但又紧密配合的部件处理，所以系统处理能力较强，吞吐率也较高。在硬件上 PE 的增加或减少很灵活，相联访问的特点是使软件编制与 PE 数量无关，PE 增减不影响软件。

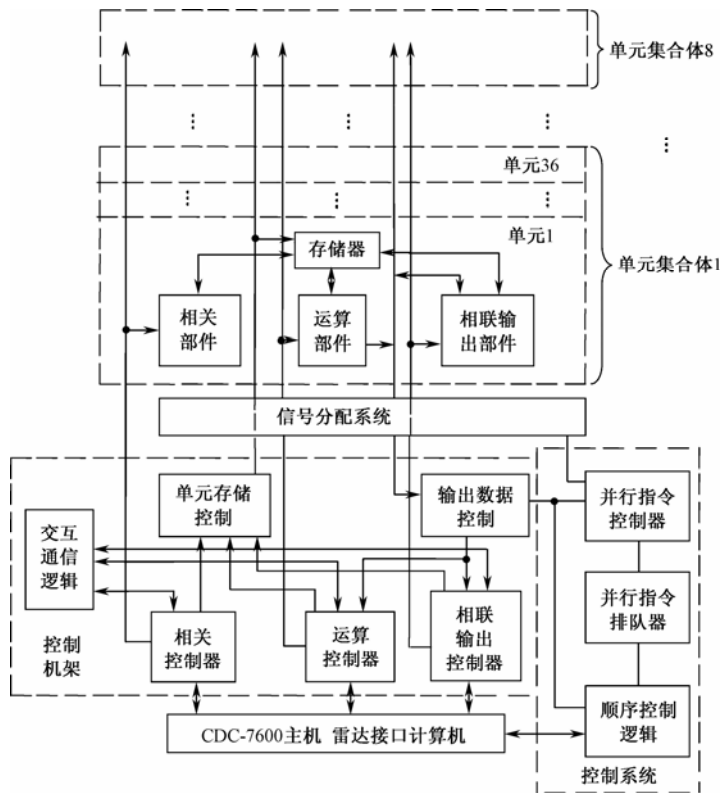


图 5-66 PEPE 系统的逻辑结构框图

2. STARAN系统

STARAN 系统由美国 Goodyear 宇航公司于 1971 年研制成功，采用“位片串字并”工作方式，其系统结构如图 5-67 所示。

接口部件右边为 STARAN 相联阵列系统，由相联阵列模块、相联控制器、程序编页器、顺序控制器、控制存储器、外部功能逻辑等组成。相联阵列模块可扩充至 32 个，主要用于实现并行的相联处理。相联控制器从相联处理控制存储器取指、交换数据、控制相联阵列模块执行有关的相联处理操作。控制存储器有 4 个 512×32 容量的双极型半导体存储器，前三个保存页面 0, 1, 2 用于存放当前执行的指令，另一个为 Cache，可存放指令和数据，通过缓冲 I/O 通道（BIO）对它进行访问。控制存储器主要用于存放相联控制程序和并行 I/O（PIO）控制程序。控制存储器有一个专门空间留给 DMA 用，通过 DMA 通道可以访问主机存储器。控制存储器中还有一个容量为 16K×32（可扩充至 32K×32），存取周期为 1μs 的磁芯存储器，用于存放主程序。存储器端口逻辑可将 0, 1, 2 页分别接至相联控制器、程序编页器和顺序控制器，而端口开关状态由外部功能逻辑控制。程序编页器是将控制程序页面从慢速磁芯存储器传送至快速半导体存储器，供执行。顺序控制器是一台小型机（如 PDP-11）用于管理外设，提供人机界面，监督系统运行。外部功能逻辑用于控制相联控制器、程序编页器、顺序

控制器相互同步。STARAN 通过接口部件与主机、外设相连，包括 DMA 通道、BIO 通道、PIO 通道、EXF(外部功能)通道。每个相联阵列模块有 256 个 I/O 端与接口部件相连。STARAN 可通过 I/O 通道与主机相连，也可直接连至主机的存储器总线，使 STARAN 直接访问主机存储器。相联阵列模块结构如图 5-68 所示，多维访问存储器容量为 256×256 ，算逻部件有 256 个 PE，还有交换互连网络 and 选择器。所谓多维访问存储器是指可按字（256 位）、按字节（8 位）、按位进行访问，由普通的 RAM 芯片构成。每个 PE 按“位串字并”方式工作。因为多维访问存储器每个字有 256 位，所以算逻部件有 256 个 PE。随着器件技术发展，多维访问存储器由 ECL 芯片构成，容量达 1024×256 ；由 MOS 芯片构成，容量达 8192×256 ，比原来大 32 倍，由于有足够存储空间存放中间结果，所以运算速度可提高 2~4 倍。

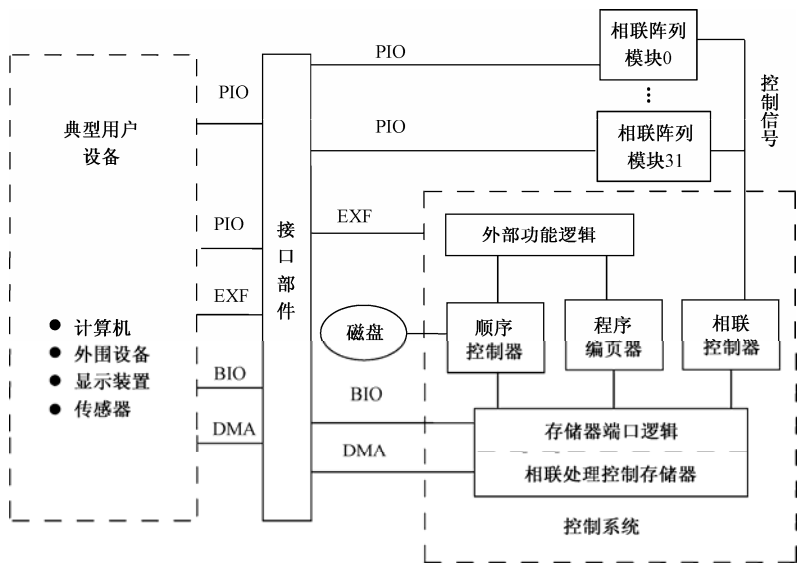


图 5-67 STARAN 机系统结构框图

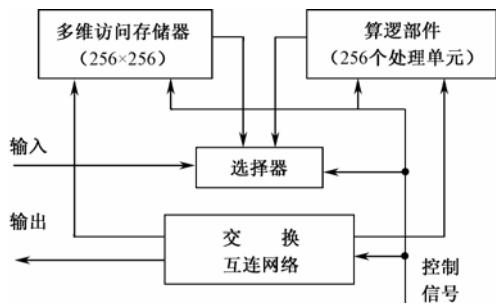


图 5-68 相联阵列模块的结构

1973 年罗马航空发展中心（RADCD）研制的 RADCAD 相联处理机系统由一台 STARAN 机、一台 Honeywell 公司造的 HIS-645 机及各种外设组成，用于空中交通管制。1974 年美国陆军地形描绘实验室（ETL）研制的相联处理机系统由一台 STARAN 机、一台 CDC6400 主机、一个具有指令通道和高速数据通道的接口部件组成，用于自动绘图、图像处理、主体摄影测量、信息检索等。1975 年 STARAN 作为美国休斯顿全国航空和宇航局约翰逊宇宙中心（JSC）专用机使用。

第 6 章 多处理机系统

多处理机属于多指令流多数据流（MIMD）计算机，可实现任务、作业级的并行。由第 4 章可知，不同的技术途径，可以发展出同构型、异构型和分布式多种处理机系统。本章重点讨论多处理机的互连技术、多处理机的系统结构、多机共享存储技术及其结构，同时也对并行处理的有关算法进行分析与探讨。

6.1 多处理机的概念

6.1.1 多处理机系统的定义

P.H.Enslow 对多处理机给出了下列定义：

- ① 包含两个或两个以上功能大致相同的处理器；
- ② 所有处理器共享一个公共内存；
- ③ 所有处理器共享 I/O 通道、控制器和外围设备；
- ④ 整个系统由统一的操作系统控制，在处理器和程序之间实现作业、任务、程序段、数组和数组元素等各级的全面并行。

既然要求实现全面并行，那么多个处理器共享资源（共享内存、I/O 通道以及外围设备）就是绝对必要的。统一操作系统是系统实现整体控制和任务调度的必要条件。所以，全面并行性是多处理机系统最根本的特征，统一操作系统是决定性因素。

多处理机具有如下明显的优点：

（1）很高的性能价格比。单处理机的性能价格比随其规模的增大而呈下降趋势，而多处理机若不考虑机间连接成本，且假定性能与处理机数目成正比，则性能价格比为一水平直线，但随着处理器数目的增加，性能价格比仍要降低。由于当前微处理器生产成本和维护费用不断下降，故多处理机性能价格比下降趋势比单机系统缓慢得多。

（2）很高的可靠性。多处理机系统有大量同构型或同功能的计算机，使其具有很大的冗余度，能在系统结构一级实现容错需求。硬件故障排除和系统维护方便，提高了系统的可维护性和可用性。

（3）很高的处理速度。单处理机加快运算速度的主要手段是提高时钟频率，但也有限制。多处理机系统凭着多个处理器并行运算的优势，可将系统的运算速度提至几千亿次/秒至几千万亿次/秒。这是单机系统所望尘莫及的。

（4）很好的模块化。多处理机系统，特别是分布式系统中，每一台处理机都可以模块化。由于超大规模集成电路（VLSI）工艺的迅猛发展，处理机甚至可以芯片级模块化（合成封装），因此可以大量重复设置，具有极好的结构灵活性、可扩充性、可重构性。这对于改善系统适应不同程序和算法时的平衡性、对额外任务和高峰负载的处理潜力，以及实现多道程序都提供了有利条件。

6.1.2 多重处理对处理机特性的要求

目前，大多数处理机系统由系列微处理器组成。同构型的微处理器有相同的特性，易于按多处理机要求构成多处理机系统。处理机在多道程序处理环境下，对其性能结构有如下要求。

1. 进程恢复能力

多处理机系统使用的处理机结构应能反映进程和处理机是两个不同的实体。如果某处理机发生故障，另一台处理机应能检测到被中断的进程状态，使被中断的进程能继续运行。没有这个功能，系统的可靠性会大大下降。大多数处理机把当前正在运行的进程状态保存在内部寄存器中，如何使其他处理器在必要时能访问到进程状态，是恢复进程的关键之一。在不太损失速度的前提下，把通用寄存器与处理机本身分开是可能的，在系统内设置所有处理机共享的通用寄存器组可以实现上述功能。

2. 有效的现场切换

需要一个共享通用寄存器组的另外一个原因是使多道程序处理机可以使用一个很大的寄存器堆。为了高效地使用它，处理机必须支持多个寻址域，因此要有域改变和现场切换操作。现场切换操作是把当前进程状态保存起来，然后通过恢复新进程的状态切换到被选中的准备好运行的进程。这种切换操作需要强化排队和堆栈操作。为了有效地进行现场切换，可以在指令系统中设置一条专门指令。该指令执行的结果是将当前进程状态或现场内容保存起来，然后到主存储器的缓冲区取另一个进程状态，该缓冲区称为交换包。这种方法如设计不当，可能使建立系统的并发操作时间开销显著增加。如果有一个大容量的寄存器组，有一个指针指向当前的寄存器组，那么改变指针内容，就可以改变选中的寄存器组，也就能有效地实现任务切换，如图 6-1 所示。当前进程寄存器就是指针。

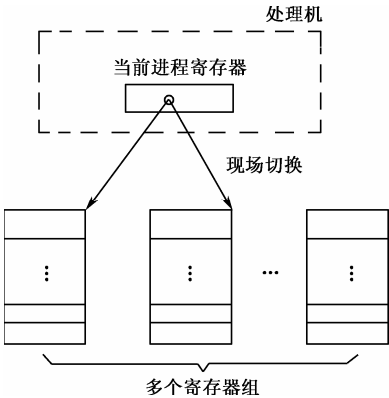


图 6-1 具有多寄存器组的处理机现场切换

3. 大的物理地址空间和虚拟地址空间

多处理机系统内的处理机必须支持大的物理地址空间（即直接寻址空间要大），这是因为进程需要访问大量数据。例如 Pentium 微处理器有 32 根地址线，直接寻址空间可达 4GB，能满足需求。有了大的物理地址空间，还需要大的虚拟地址空间，把虚拟地址空间分段，便于模块共享以及地址界限的检查。

4. 高效率的同步原语

处理机设计时必须提供作为同步原语基础的某种不可再分的操作。这些同步原语需要有互斥机构支持。当两个以上的进程并发地运行或相互交换数据时，需要互斥。互斥机构包含某种形式的“读—修改—写”存储周期和排队。信号灯（Semaphore）是互斥机构的一种。每个信号灯有其队列，队列中的项是被挂起来的进程。信号灯操作是不可分操作，利用“读—修改—写”存储周期，测试和修改信号灯。队列操作也应是不可分的。

5. 处理机之间有高效率的通信机构

构成多处理机系统的处理机必须具有高效率的通信机构。通信机构一般用硬件实现，这有助于实现处理机之间的同步。在非对称多处理机系统中，不同处理机之间经常需要交换服务请求，硬件通信机构作用更加明显。在处理机发生故障时，通过该机构发信号给其他正在运行的处理机，并启动诊断或纠错过程。在紧密耦合的多处理机系统内有共享存储器，采用软件方法实现多处理机之间的通信是可能的。每个处理机必须周期性地查询位于共享存储器内的“信箱（缓冲区）”，检查是否有信息给它。这种方法效率较低，当处理机数目很大时，查询时间就很长，而且需对多个查询请求予以仲裁。因此，必须有一个或若干处理机等待，这就要求处理机内有等待状态的机构，或将该处理机在队列中挂起的机构。

6. 指令系统

处理机指令系统应支持实现过程级并发功能的高级语言，为有效地处理数据结构提供充分条件。所以指令系统内应有过程连接、循环结构、参数处理、多维下标计算和地址界限检查等指令。此外，还需包括产生和结束程序内部并行执行通路的指令。因此，指令系统有丰富而齐全的寻址方式。为了产生一个唯一的进程标识名和进程管理需要的超时信号，处理机应具有硬件计数器和实时钟。这些定时器在多重处理机系统中还可用于检测各种错误，如将“看门狗”定时器与重要的系统资源联系起来。当定时器在指定时间内不清零时，则以某种方式（如发中断请求信号）产生一个出错信号。

6.2 多处理机结构

6.2.1 多处理机的基本结构

下面从多处理机系统常用的松散耦合和紧密耦合两种形式来看它们的基本结构。

1. 松散耦合多处理机结构

这种形式的结构互连常用通道或通信线路来实现，它们连接的频带较低。现举两例说明。

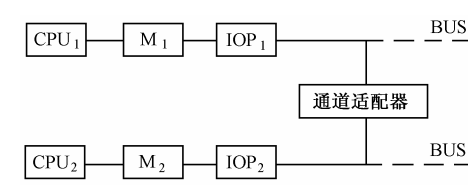


图 6-2 用通道互连的松散耦合系统

(1) 通道连接的多处理机结构。如图 6-2 所示是用通道互连的松散耦合的系统。每台计算机是独立的，它们之间通过通道适配器连接。在进行通信时，发送方计算机可以视接收方计算机为自己的一个 I/O 设备，从而完成两个主存储器之间的数据传输。

(2) 信息传输系统连接的多处理机结构。如图 6-3 所示，是多个计算机模块通过一个信息传输系统连接起来的松散耦合系统。信息传输系统耦合程度较低，常用简单的分时总线及环形、星形等拓扑结构。每个计算机模块可以是独立的计算机，它有处理单元、存储器、I/O 部件。而模块与信息传输系统则通过通道仲裁开关相连。通道仲裁开关的作用除了使彼此通信的计算机模块在信息传输系统里连接起来外，还在多个模块同时申请信息传输时，决定本模块是提出申请还是延缓提出申请，故称其有仲裁作用。

2. 紧密耦合多处理机结构

紧密耦合多处理机结构往往是高频带的，通常由高速总线或高速开关实现机间互连，以

便共享存储器。如图 6-4 所示，是一种比较典型的紧密耦合系统。

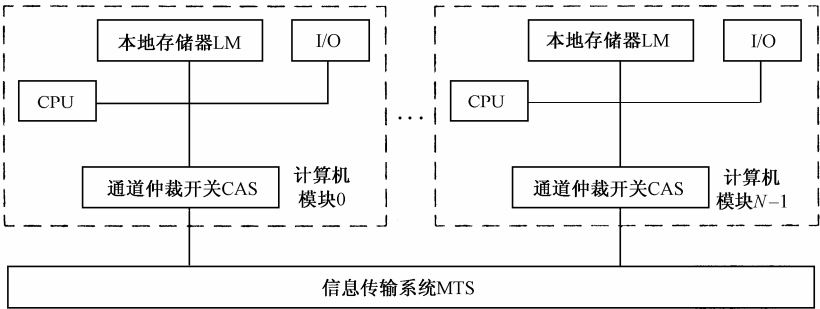


图 6-3 用信息传输系统互连的松散耦合系统

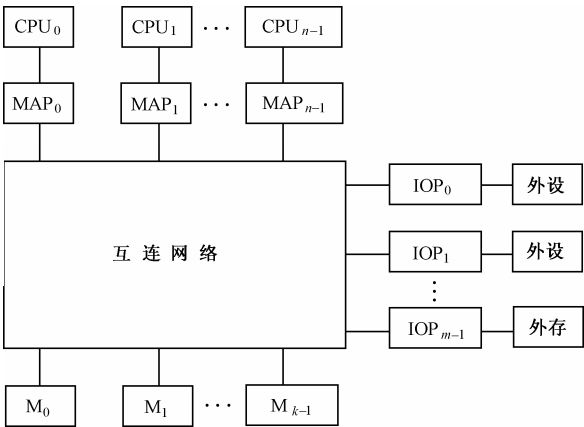


图 6-4 用互连网络连接的多处理机系统

多个处理器通过互连网络（由高速开关组成）共享集中的主存储器（它由若干存储模块组成）和多个输入/输出设备。当某个处理机要访问主存储器时，只需通过它的存储映像部件（MAP），就可以把全局的逻辑地址转换成局部的物理地址（某一存储模块内的物理地址）。互连网络不仅提供高速传输通路，而且具有选择有效路径、仲裁访问冲突等功能。对于输入/输出设备的访问也和访问存储器一样，只是它们的界面通过输入/输出处理机（IOP）进行。

6.2.2 多处理机的互连网络

多处理机的主要特点是，各台处理机共享一组存储器和 I/O 设备。这种共享功能是通过两个互连网络实现的：一个是处理机和存储器模块之间的互连网络；另一个是处理机和 I/O 子系统（I/O 接口和 I/O 设备）之间的互连网络。互连网络可以采用不同的物理形式，一般可有下列 4 种基本结构。

1. 总线结构

多处理机结构最简单的互连系统是把所有功能模块（或部件）连接到一条公共通信通路上，如图 6-5 所示。公共通信通路也称时分或公共总线。它的特点是简单、易实现，也容易扩展（重构）。总线是一个无源部件，通信完全由发送和接收的总线接口控制。由于总线是共享资源，所以必须有总线请求和仲裁机构，以避免发生总线冲突。

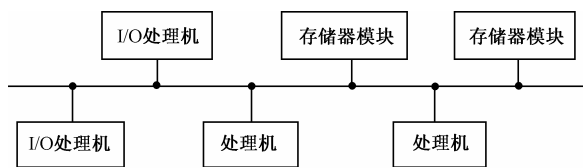


图 6-5 单总线的多处理机组织

总线仲裁方法有静态或动态的优先级方法、先进先出（FIFO）队列方法、串行优先链方法和总线控制器（或仲裁器）方法。当一个处理机要占用总线时，首先需测试总线状态是否“忙（Busy）”。若是忙，则等待；等到空闲（即不“忙”）时，发出总线请求信号，经仲裁后，得到总线响应信号，才可以占用总线，与目的部件进行通信。在一个处理机占用总线进行通信的过程中，哪怕比其优先级高的处理机需占用总线，也不能终止（中断）正在进行的通信过程。

单总线结构简易可靠，但总线接口线路出现任何一个故障都会造成系统的瘫痪。另外，如果系统扩展更多的处理机或存储器，将会增加总线竞争，从而增加仲裁逻辑，降低系统吞吐率。系统内总线通信速率也受到单总线带宽和速度的限制。所以若各处理机有自用存储器和自用 I/O 设备，可减少总线上的通信负荷，但会增加系统互连的复杂性。

为了提高总线通信效率，可设置两条单向通路，如图 6-6 所示。一个传输操作用到两条总线（输入/输出），因此实际上收益不大。另一个是设置多条双向总线，如图 6-7 所示，在同一时间可进行多条总线通信，但会增加系统的复杂性。

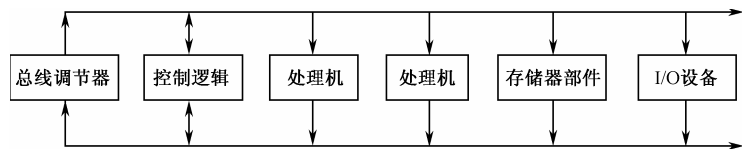


图 6-6 具有两条单总线的多处理机结构

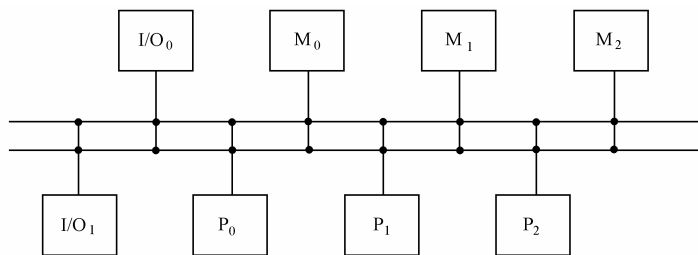


图 6-7 多总线的多处理机结构

影响总线性能的因素有：总线上主控设备（即能掌握、占用总线的部件）的数量、总线仲裁算法、控制集中程度、数据宽度、数据传输同步和错误检测等。

总线仲裁常用硬件实现，并允许在一个总线通信过程内对下一个总线请求予以仲裁。总线仲裁算法有下列 4 种。

（1）静态优先级算法。给每一个设备一个唯一的优先级。当多个设备同时提出请求时，其中优先级别最高者占用总线。常用串行优先链（也称菊花链）方式实现，每个设备的优先

级以其在链上的物理位置决定，距总线控制器越近，则其优先级越高，如图 6-8 所示。也可将各设备的总线请求信号送至“编码-译码逻辑”，各设备均有各自的编码器输入端，而编码器本身对各输入端信号予以排队，同时有几个请求信号进入编码器，由编码器按其已确定的优先级，选择其中优先级别最高者的编码送译码器，然后给出相应信号使级别最高设备占用总线。这种方式也称并行仲裁方式，如图 6-9 所示。

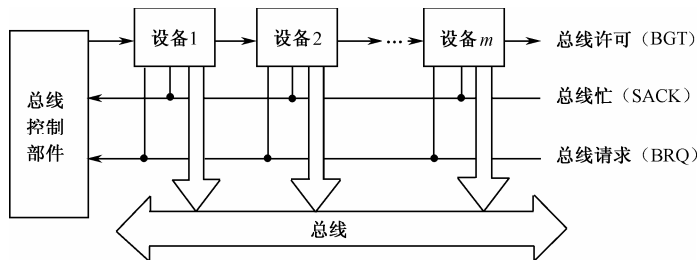


图 6-8 总线结构的串行仲裁方式

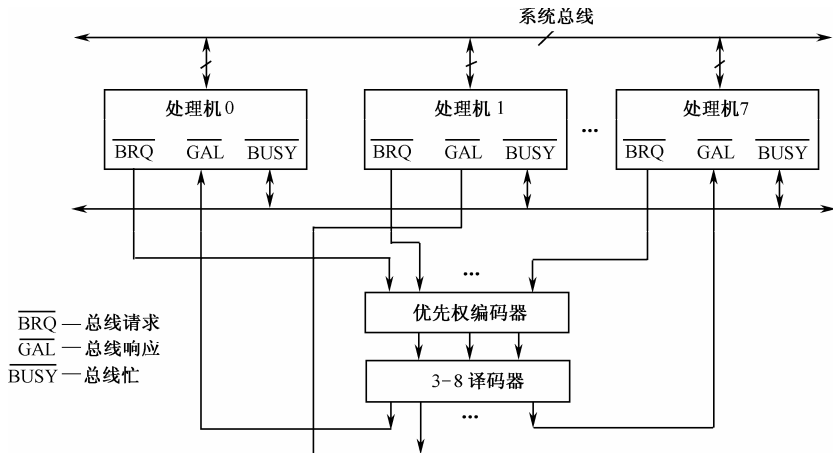


图 6-9 总线结构的并行仲裁方式

（2）固定时间片算法。把总线可用的带宽分成固定长度的时间片，然后把时间片按循环方式顺序分配给每个设备。如果被选中的设备不使用它的时间片，那么其他设备也不能占用这个时间片。这种技术称为固定时间片 FTS（Fixed Time Slicing）或分时多路转接 TDM（Time Division Multiplexing）。采用 FTS 方法时，各个设备（处理机、主控模块等）的级别是相同的，无先后之分。按照这种算法，某一设备最长等待时间不会超过定值，而平均等待时间比较长，因此总线利用率比较低；在总线负载较轻时，有较高的性能。它的简单性使它在要求不高的场合得到了比较广泛的应用。

（3）动态优先级算法。对每个设备的优先级予以动态调整，使每个设备均有机会占用总线。动态改变优先级的算法有两种：近期最少使用 LRU（Least Recently Used）算法和旋转菊花链 RDC（Rotating Daisy Chain）算法。

LRU 算法是把最高优先级分配给最长时间没有占用总线的但已提出请求的设备。实现方法是，在每个总线周期后重新分配优先级。RDC 算法的结构中无总线控制器，将总线许可线环接起来，如图 6-10 所示。当某一设备已经被许可占用总线时，它便作为下次仲裁的总线控

制器，紧随其后的一个设备就成为下一次仲裁时串行链上优先级最高的设备。因此，每一个总线周期后优先级都会动态地改变一次。

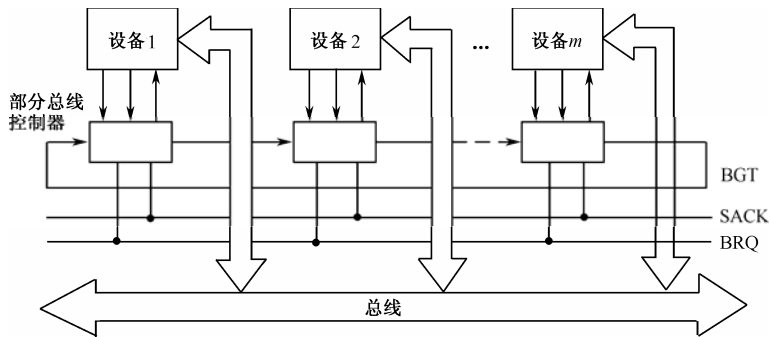


图 6-10 RDC 算法结构

(4) 先来先服务算法。先来先服务 FCFS (First-Come First-Served) 算法是按照接收到请求的先后顺序予以处理的。如果总线传输时间固定，那么 FCFS 具有最小平均等待时间。就性能而言，FCFS 是总线最佳仲裁算法。但 FCFS 实现困难，一个原因是必须有一个逻辑来记录所有请求的次序。另一个原因是当第二个请求信号到达的时间间隔很小时，无法准确地区分它们的先后。因此，FCFS 只是一个近似的解决方法。

2. 交叉开关

当不断增加总线数目时，使得每个存储器模块都有它自己单独可用的通路，如图 6-11 所示。这种互连网络称为无阻塞交叉开关 (Nonblocking Crossbar)。由于每个存储器存储模块有其自己的总线，所以交叉开关实现了存储器模块的全连接。因此，可同时进行通信的开关个数受总线的带宽、速度以及存储器数目的限制但不受通路数的限制。交叉开关的特点是开关和功能部件的接口非常简单，而且支持所有存储器模块同时通信。每个交叉点不仅能切换并行传播，而且还必须能解决在同一存储器周期内访问同一个存储器模块的多个请求之间的冲突，通常用预设的优先级来处理冲突，这就导致了开关硬件非常复杂。

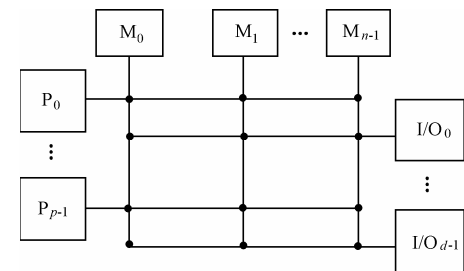


图 6-11 交叉开关的结构

交叉开关网络中的交叉结点结构如图 6-12 所示。该网络以 16 个目的部件 (存储器模块) 和 16 个主控模块 (处理机) 为例。多路转接器模块工作过程如下：根据仲裁模块的决定，从 16 路中选中某一路，使该路的数据线、地址线、读/写控制线与存储器模块接通，使该路的处理机与存储器模块进行通信。而仲裁模块接收 16 路的请求 (REQ₀~REQ₁₅)，依照优先级次序，通过优先权编码器，将选中的最优者编码送多路转接器，同时回送相应的响应信号 (ACK)。

交叉开关结构也可推广到 I/O 设备上，在 I/O 处理机的另一边使用类似的开关，实现 I/O 设备的交叉连接，如图 6-13 所示。

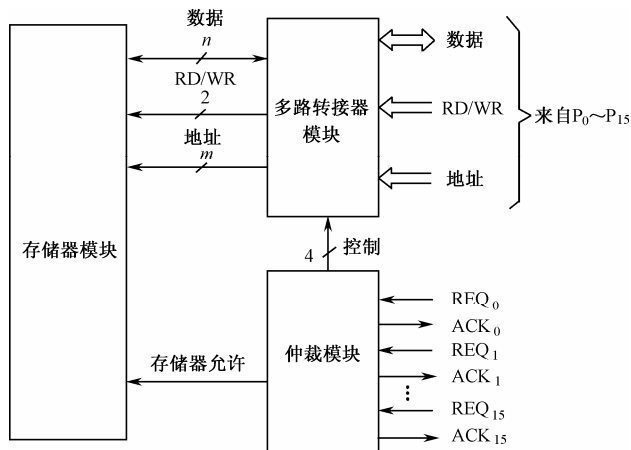


图 6-12 交叉开关结点的结构

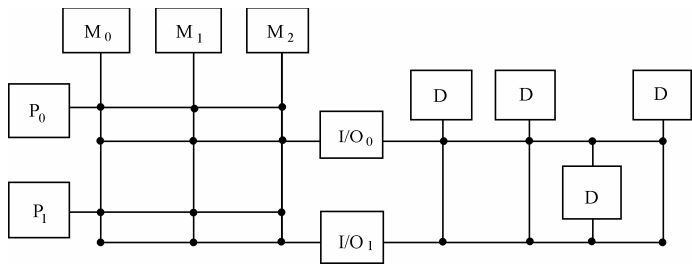


图 6-13 处理机-存储器-I/O 互连交叉开关

3. 多端口存储器

如果把分布在交叉开关矩阵网络上的控制、转接、优先级仲裁等逻辑功能转移到存储器模块的接口上，就形成了多端口存储器系统，如图 6-14 所示。这种系统既适合于单处理机，也适合于多处理机。对于访问存储器的冲突，常用的解决方法是，每个存储器端口分配一个永久优先级，而各个主控模块相对于某个存储器模块有一个优先级别序列。例如，对于 M_0 而言，其能接收主控模块的访问优先次序为 $P_0, P_1, I/O_0, I/O_1$ ；对于 M_1 而言，则为 $P_0, P_1, I/O_1, I/O_0$ ；对于 M_2 而言，则为 $P_1, P_0, I/O_0, I/O_1$ ；对于 M_3 而言，则为 $P_1, P_0, I/O_1, I/O_0$ 。

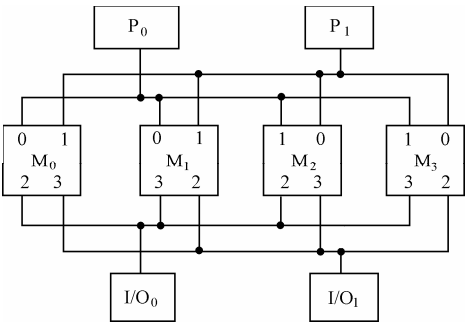


图 6-14 具有优先级的多端口存储器系统

4. 多处理机的多级网络

由于开关过于复杂，大规模交叉开关往往使用多个小规模交叉开关“串联”和“并联”组成多级交叉开关网络，以取代单级的大规模交叉开关。图 6-15 所示是用 4×4 交叉开关模块组成的二级 16×16 的交叉开关网络，其开关数量比单级 16×16 交叉开关网络节省一半。在图 6-15 中， $4^2 \times 4^2$ 表示如下： 4×4 是交叉开关基本模块，指数 2 为互连网络级数。将其扩展：以 $a \times b$ 交叉开关模块为基础，可构成 $a^n \times b^n$ 的开关网络，这已为 Patel (1981) 多处理机采用，称为 Delta 网络。Delta 网络的级间互连用典型的分组混选实现。如有 $N = qc$ 张牌，分组混选是将 qc 张牌分成 q 组，每组 c 张，依次取各组的第 1 张，然后依次取各组的第 2 张，依次类推，共取 c 次。图 6-16 为 12 个结点的 4 组混选，其中， $q = 4, c = 3$ ，记作 $S_{4 \times 3}$ 。

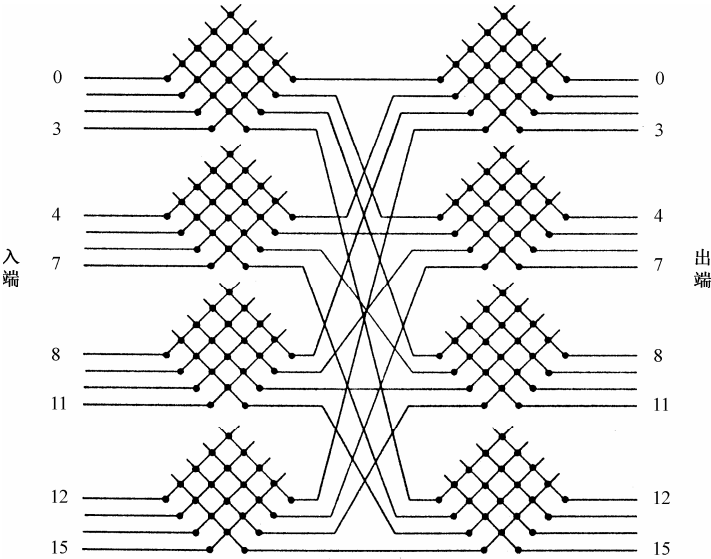
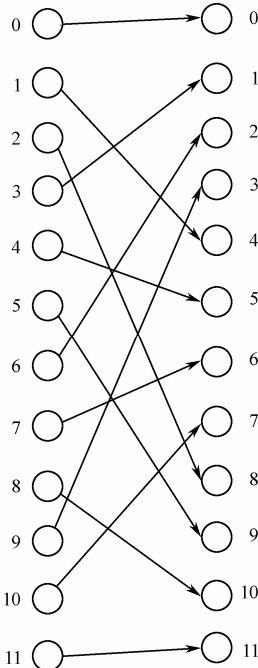


图 6-15 $4^2 \times 4^2$ 交叉开关



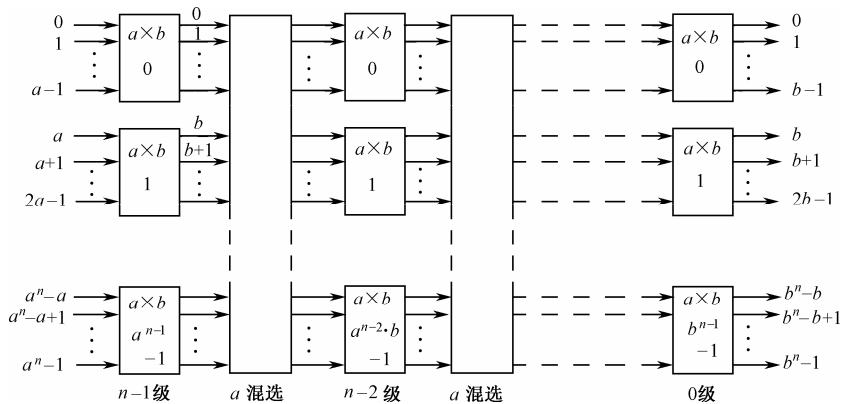


图 6-17 n 级 $a^n \times b^n$ 的 Delta 网络

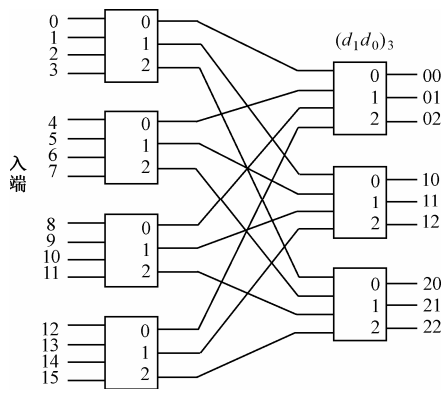


图 6-18 $4^2 \times 3^2$ 的 Delta 网络

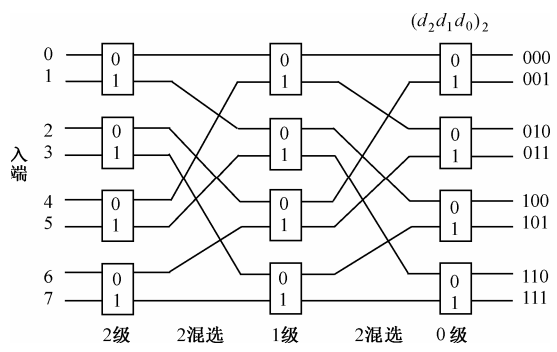


图 6-19 $2^3 \times 2^3$ 的 Delta 网络

多处理机的多级网络的基础是交叉开关。所以互连方式以前三种结构为基本类别，即总线结构、交叉开关、多端口寄存器。这三种互连方式的优缺点如下。

采用总线结构的多处理机系统：① 系统硬件成本最低且简单；② 通过增、删功能模块（部件）可方便地改变系统硬件配置；③ 总线通信速率限制了整个系统的容量，总线故障会使整个系统瘫痪；④ 系统以增加功能模块方式进行扩充会降低整个系统的吞吐率；⑤ 三种基本互连方式中本方式系统效率最低；⑥ 本方式适用于规模较小的系统。

采用交叉开关系统的多处理机系统：① 互连系统最复杂，潜在的总通信速率最高；② 因为控制和切换逻辑在开关内部，所以功能模块最简单且最便宜；③ 因为任何功能模块要装配到系统中都需要使用一个基本开关矩阵，所以本方式只面向多处理机才能使性能价格比趋于合理；④ 系统扩充（增加功能部件）会提高整个系统性能，而且不必重写操作系统，因此有最高的系统潜在效率；⑤ 从理论上讲，系统扩充只受开关矩阵大小的限制，而开关矩阵的设计和制造可采用模块化方式予以扩展；⑥ 开关内部采用与/或的冗余方法提高了开关的可靠性，因此也就提高了系统的可靠性。

采用多端口存储器的多处理机系统：① 因为大多数控制和切换逻辑在存储器模块中，所以存储器模块价格最高；② 由于本方式对功能模块特性要求不高，因此可使用较低档的处理机；③ 整个系统有很高的潜在的总通信速率；④ 在系统设计早期就应对存储器端口数和

类型予以确定，因而也就确定了系统的规模和配置，确定后很难修改；⑤ 需要大量的连接设备。

6.2.3 多处理机系统的存储器结构

1. 主存的组成

在多处理机系统中，为了减少访存冲突，主存采用并行存储器结构。多个存储模块可采用低位交叉编址技术，也可采用高位交叉编址技术。如果多处理机的已激活的进程共享集中的地址空间，则低位交叉编址有其优越性。反之，如果共享的地址空间很小，则低位交叉反而会引起存储器冲突，而高位交叉编址可为某进程集中一定数量的页面，可以有效地减少存

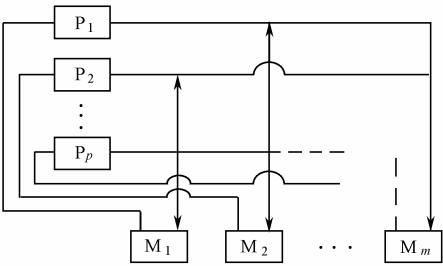


图 6-20 宿主存储器结构

储冲突。能为某处理机进程放置大多数页面的存储器模块称为该处理机的宿主存储器。如果该处理器的现行进程全部活动页面在宿主存储器内，而且该存储器不包含其他处理机的页面，则处理机不会遇到存储冲突。这种结构如图 6-20 所示。每个存储模块有两个端口，一个连到互连网络，另一个直接连到相应的处理机。由于处理机可直接访问宿主存储器，减少了通过互连网络访存的延迟时间。

当每个处理机有其自己的 Cache 时，如果采用低位交叉编址，则 Cache 内某一块信息的各存储单元可能降落在主存各个存储模块内。这样，该信息块在传输时，需要对互连网络进行频繁转接，从而使通信效率严重下降。因此，多处理机系统中常采用二维存储器结构，如图 6-21 所示。在这个主存中有 N 个 ($N = 2^n$) 同样容量的存储模块，排成 L 列 (体)，每一列由 m 个模块组成。各列之间按高位交叉编址，而列内各模块按低位交叉编址，每列有一个列控制器连到互连网络。当访问 Cache 不命中时，需访问主存进行一次块传输。一块信息通常驻留在某列中。因此，当列控制器 LC 接收到传输长度为 b ($b \leq m$) 的信息块请求时，LC 对本列发出 b 个内部请求，对 b 个相邻模块予以访问。

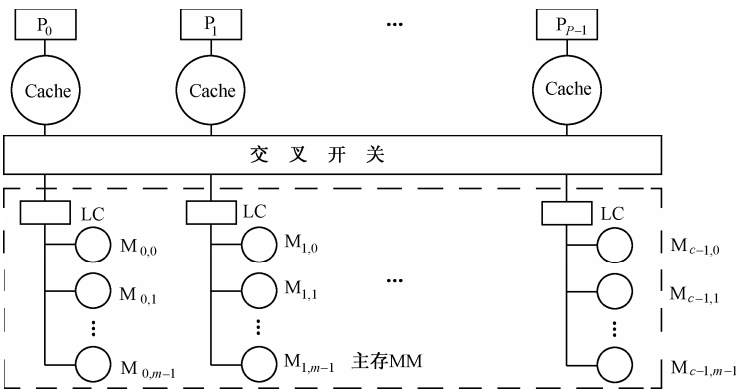


图 6-21 二维存储器结构

2. 多处理机系统的Cache结构

当每个处理机都有自己专用的 Cache 时，对应主存中某一个单元的数据，在各个 Cache

中可能会出现相应的副本。当对其中某一个副本进行一次修改操作时，会产生 Cache 中数据的不一致性。无论 Cache 采用“写回法”或“写直达法”，都不能解决多个 Cache 不一致的问题。例如，采用写直接法，处理机 A 进行写操作，可以保证它的 Cache 和主存内容一致。而处理机 B 的 Cache 可能有主存同一单元的副本，并没有得到相应的修改而出现不一致性。尤其是在多道程序的多处理机系统中，如果被挂起的进程以后又移到其他处理机上继续执行，由于该进程最近修改过的数据可能仍留在原处理机的 Cache 中，而新处理机上该进程仍在使用原来数据（即新处理机 Cache 中有关数据来不及修改），此时，进程的现场恢复就会出现错误，从而导致系统工作出错。解决 Cache 的一致性问题有如下两种方法。

（1）静态一致性校验。其基本思想是，只让该进程的独用信息（指令和操作数据）和共享的只读信息进入本处理机的 Cache，而对于共享的可写（即可修改）的信息不准进入 Cache，只可留在主存中。若 s 为访问共享可写信息的概率， t_M 为主存取周期， t_C 为 Cache 存取周期，则平均存取周期为

$$(1-s)t_C + st_M = t_C \left[(1-s) + s \cdot \frac{t_M}{t_C} \right]$$

显然，当 t_M/t_C 很大时，说明访问共享主存时间大大增加，加剧了互连网络和主存的竞争，因此性能较差。减少竞争的方法是增加一个共享 Cache，即 SC (Shared Cache)。共享信息均在 SC 内，而取指令和独用数据则通过独用 Cache，即 PC (Private Cache)，其结构如图 6-22 所示。SC 也可以由多个模块构成，通过互连网络连到处理机和共享存储器上。这个网络的复杂性比交叉开关低，如果各种 Cache 命中率均很高，则能减少对主存的竞争。SC 要求能识别访问主存的信息是独用的还是共享的，可利用模块结构语言（如并发 PASCAL）在编译时完成。但共享 Cache (SC) 结构缺乏灵活性。

（2）动态一致性校验。基本思想是，在若干 Cache 中使同一个信息（指令、数据）始终保持动态一致。

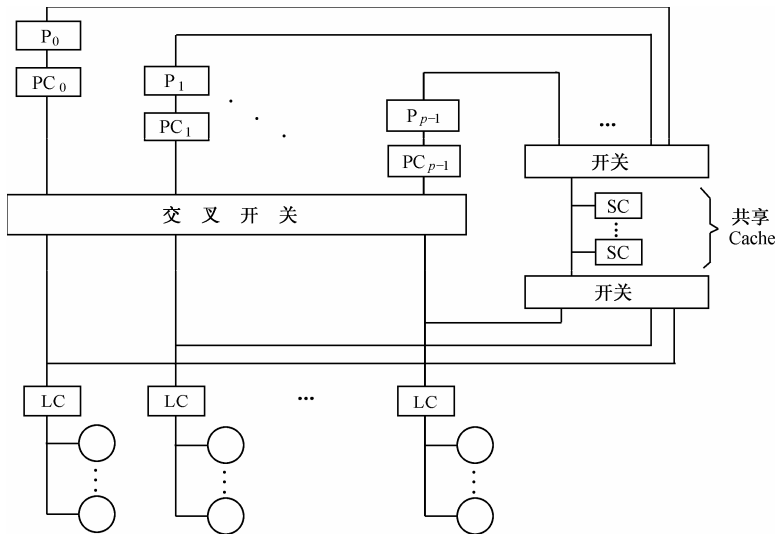


图 6-22 具有共享 Cache 的多处理机结构

一种方法是广播法。即当每个处理机每次写 Cache 时，不仅写入自己的 Cache 和共享的主存中，而且还把信息送到所有 Cache，如果其他 Cache 有与自己 Cache 相同的目标行，则

也进行改写。显然，这种方法增加了传输的信息量。为此，可采用作废法，即某个处理机写 Cache 时，不但同时改写主存的有关单元，而且发一个信号到所有其他 Cache，通知“某个存储单元内容已修改”。如果其他 Cache 中有包括该单元的块，则使之作废，当处理机再次访问此块时，就从主存调入已修改过的块。为了保证执行的正确性，发出信号的处理机必须等待，当收到其他处理机的应答信号后，它才能进行写入主存的操作。

另一种方法是目录法。在快速 RAM 中构建一个目录表，如图 6-23 所示。它有两个部分：①存在表（Present Table），它是二维的，其中每一项 $P(i,c)$ 表示第 i 块是在第 c 个 Cache 中；②修改表（Modified Table），它是一维的，其中每项 $M(i)$ 表示第 i 块是否被修改过。在每个 Cache 中还有一个本地标志（可在地址变换表中设立） $L(k,c)$ ，表示第 c 个 Cache 中块 k 的状态。它可有三种状态。

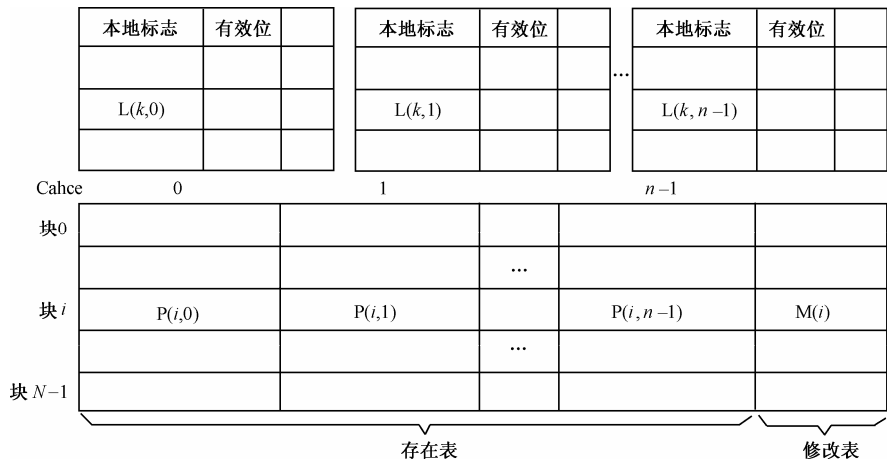


图 6-23 用于动态一致性检验的目录表

- ① RO 状态。该块信息有多个副本，任何副本均没有修改过。
- ② EX 状态。该块信息仅有一个副本，且该副本未修改。
- ③ RW 状态。该块信息仅有一个副本，且该副本已修改。

为了保证 Cache 的一致性，在任一时刻仅允许一个处理机对某块信息拥有 EX 或 RW 的副本。图 6-24 和图 6-25 是处理机在本地 Cache 中读和写一块信息的流程图。从中可以看出，如何保证该块信息在多个 Cache 中的状态仅有一个 Cache 被置成 EX 或 RW 状态。该流程是按“写回法”且按写分配的条件设计的。

6.2.4 多处理机系统的特点

在多处理机系统中，互连网络是决定其性能的重要因素。此外，还应了解多处理机系统的其他一些特点。

(1) 结构灵活性。相比并行处理机的专用性，多处理机系统把能并行处理的任务、数组，以及标量都进行并行处理，有较强的通用性。因此多处理机系统要能适应多样化的算法，具有更灵活的结构，以实现各种复杂的机间互连模式。

(2) 程序并行性。在多处理机中，并行性存在于指令外部，即表现在多任务之间。为充分发挥系统通用性的优点，便于利用多种途径：算法、程序语言、编译、操作系统以至指令、硬件等，尽量挖掘各种潜在的并行性。

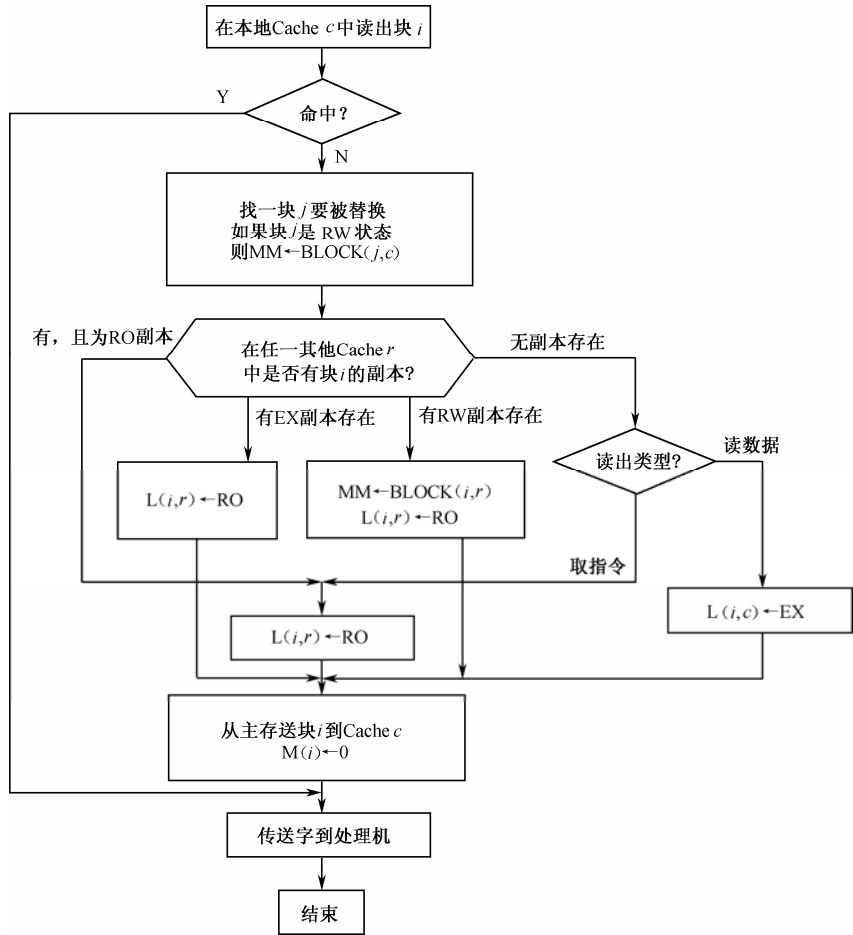


图 6-24 读 Cache 的操作流程图

(3) 并行任务派生。多处理机是多指令流操作方式，一个程序当中存在多个并发的程序段，需要专门的指令来表示它们的并发关系，以及控制它们的并发执行，使一个任务正在执行时就能派生可与它并行执行的另一些任务。

(4) 进程同步。在多处理机系统里，同一时刻，不同的处理机执行不同的指令。由于执行时间互不相等，它们的工作进度不会也不必保持相同。因此当并发程序之间有数据交往或控制依赖时，要采取特殊的同步措施，使它们包含的指令相互间仍保持程序要求的正确顺序。

(5) 资源分配和任务调度。多处理机执行并发任务时，对处理机的数目没有固定要求，各个处理机进入或退出任务和所需资源的变化情况都很复杂，因此资源分配和任务调度的好坏将直接影响整个系统的效率。

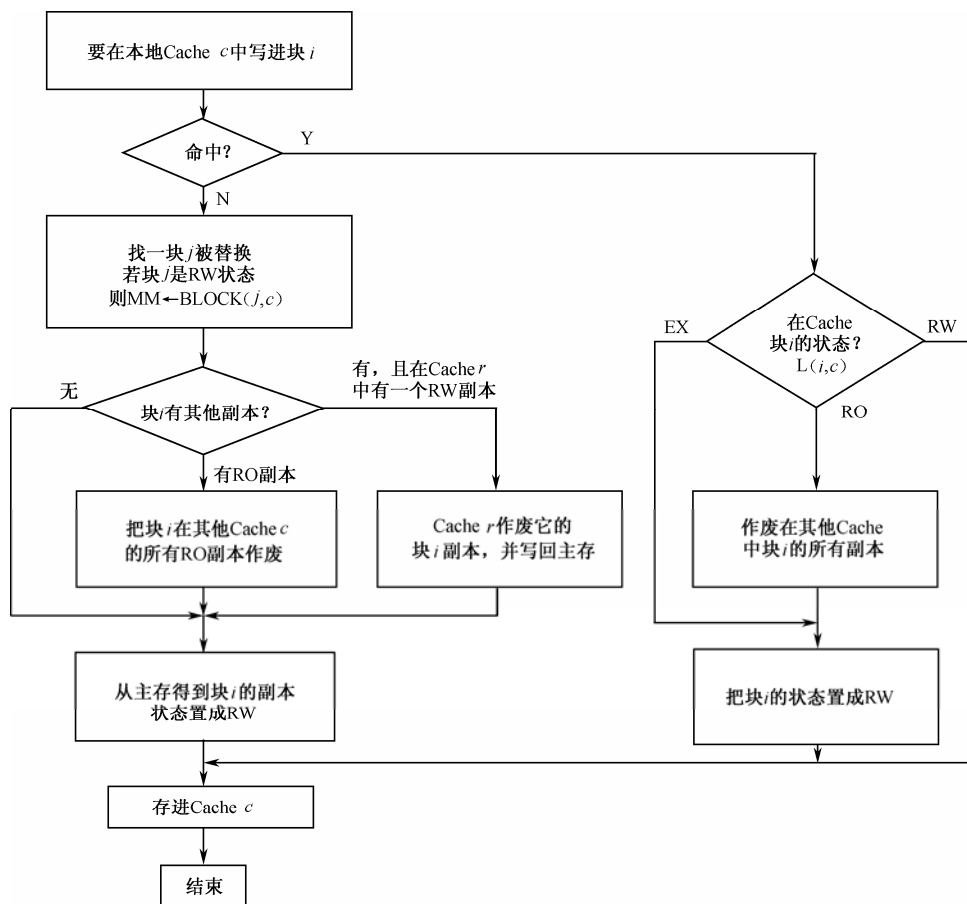


图 6-25 写 Cache 的操作流程图

6.3 多处理机的软件

6.3.1 算术表达式的并行算法

并行性的开发在于算法。顺序处理机习惯采用的循环及迭代算法，往往不适用于多处理机。而采用直解法，有时能揭示更多的并行性。例如，下列多项式

$$E_1 = a + bx + cx^2 + dx^3 \tag{1}$$

利用 Horner 法则，可得到 $E_1 = a + x[b + x(c + xd)]$ (2)

这是顺序处理的典型算法，共需三个“乘-加”循环，6 级运算，如图 6-26 (b) 所示。它对于多处理机并不合适，而采用 (1) 式算法更加有效，只需 4 级运算，如图 6-26 (a) 所示。图中 P 为所需处理机数目； T_p 为运算级数； S_p 为加速度， $S_p = T_1/T_p$ ； E_p 为效率， $E_p = S_p/P$ 。可见， $S_p > 1$ ，而 $E_p < 1$ ，即运算的加速总是伴随着效率的降低。

为了达到算术表达式并行运算的目的，常用交换律、结合律和分配律将其适当变形，以利于并行算法的建立。从算术表达式最直接的形式出发，利用交换律把相同运算集中在一起，再利用结合律把参加运算的操作数（称原子）配对，尽可能并行运算，最后利用分配律，平衡各分支运算的级数，使总级数减至最少。例如，某多项式

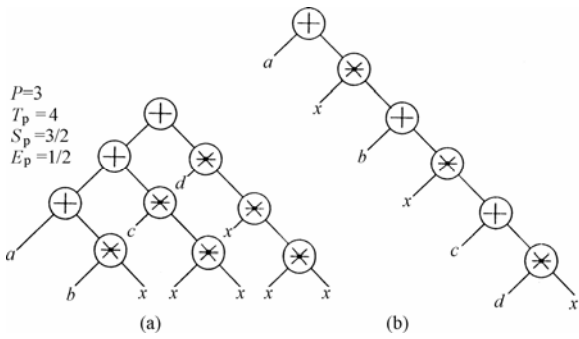


图 6-26 顺序算法与并行算法比较

$E_2 = a + b(c + def + g) + h$
需要 7 级运算。利用交换律和结合律改写为
 $E_2 = (a + h) + b[(c + g) + def]$
则需 5 级运算。再利用分配律，将上式改写为
 $E_2 = (a + h) + (bc + bg) + bdef$
则仅需 4 级运算，如图 6-27 所示。

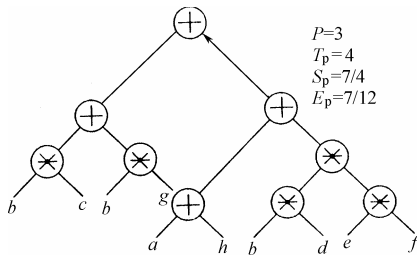


图 6-27 利用交换律、分配律和结合律的并行算法

6.3.2 程序并行性分析

并行性的关键在算法。研究并行算法及程序设计，一直是多处理机系统的一个重要研究课题。这里着重对程序并行性做一些粗略的分析。

假设有一个程序包含 $P_1, P_2, P_3, \dots, P_i, \dots, P_j, \dots, P_n$ 等多个程序段，按照书写格式当然是串行的。为了讨论方便，取最简单的形式： P_i 和 P_j 都是一条语句， P_i 在 P_j 之前执行，且只讨论 P_i 和 P_j 的直接数据相关关系。一般而言， P_i 和 P_j 之间存在三种可能的数据相关情况。

(1) 如果 P_i 的左部变量也在 P_j 的右部变量集内，且 P_j 要从 P_i 取得算出的值，则称 P_j “数据相关”于 P_i 。如

$$\begin{aligned} P_1 \quad & A = B + D \\ P_2 \quad & C = A * E \end{aligned}$$

P_2 必须取 P_1 算得的 A 值作为操作数。

(2) 如果 P_j 的左部变量也在 P_i 的右部变量集内，则称 P_i “数据反相关”于 P_j 。如

$$\begin{aligned} P_1 \quad & C = A * E \\ P_2 \quad & A = B + D \end{aligned}$$

在 P_1 未取用 A 值之前， A 的值不能被 P_2 所改变。

(3) 如果 P_i 的左部变量也在 P_j 的左部变量，则称 P_j “数据输出相关”于 P_i 。如

$$\begin{aligned} P_1 \quad & A = B + D \\ P_2 \quad & A = A + C \end{aligned}$$

P_2 存入它自己算得的值必须在 P_1 存入之后。

除了这三种情况之外，便是 P_i 和 P_j 数据不相关。这三种数据相关情况对程序并行性的影响表现为下列几种可能的执行次序。

(1) 写-读串行次序。如果程序必须保持在先的语句先写，在后的语句后读的次序，则允

许存在上述任意一种数据相关情形。一般情况下，执行的次序不可并行，也不可颠倒。这是通常遇到的典型串行程序。但有一种特殊情况，即当 P_1 和 P_2 服从交换律时，虽仍需串行执行，但允许 P_1 和 P_2 执行的次序对换，这称为可交换串行。如

$$\begin{aligned} P_1 \quad A &= 2 * A \\ P_2 \quad A &= 3 * A \\ P_1 \quad A &= B + 1 \\ P_2 \quad B &= A + 1 \end{aligned}$$

为可交换串行；但

就是不可交换串行的。

（2）读-写次序。如果两个程序段之间只包含第（2）种数据相关情形，则只须保持在先的语句先读，在后的语句后写的次序，便能既允许它们串行，也允许它们并行执行，但不允许交换次序。如

$$\begin{aligned} P_1 \quad A &= B + D \\ P_2 \quad B &= C + E \end{aligned}$$

二者同时执行，能保证 P_1 先读 B ， P_2 后写 B 的次序。又如

$$\begin{aligned} P_1 \quad A &= 3 * C / B \\ P_2 \quad B &= C * 5 \\ P_3 \quad C &= 7 + E \end{aligned}$$

虽属可能，但由于处理时间上 P_1 费时最长， P_2 次之， P_3 最短，如果不采取特别的同步措施，就不能保证必要的先读后写次序。

（3）可并行次序。如果程序的两段之间不存在任何一种数据相关情况，即无共同变量或共同变量都在右边部分，且不在左边部分出现，则二者可以无条件地并行执行。当然，也可以串行执行，而串行时可以选择任一先后顺序。如

$$\begin{aligned} P_1 \quad A &= B + C \\ P_2 \quad D &= B + E \end{aligned}$$

（4）必并行次序。如果两程序段的输入变量互为输出变量，则必须并行执行，而不允许串行执行。如

$$\begin{aligned} P_1 \quad A &= B \\ P_2 \quad B &= A \end{aligned}$$

两语句的左右变量互相交换，必须并行执行，且需保持读、写完全同步。

6.3.3 并行程序语言

程序的并行性被开发后，需要在源程序和目标程序中有专门的语句和指令，以便具体描述这些并行关系。

1. 高级语言中描述并行性的语句

多处理机的高级语言可以是原高级语言的扩展，或专门设计的新语言。但是它们都必须含有若干种描述程序并行性的语句。例如，E.W.Dijkstra 的语言方案是块结构语言的发展，它把所有可并行执行的语句或进程 S_1, S_2, \dots, S_n 用 Cobegin-Coend（或 Parbegin-Parend）括起来，如下例：

```
begin
    S0;
Cobegin
```

```
S1;  
S2;  
⋮  
Sn;  
Coend  
Sn+1;  
end
```

本程序规定，在执行语句 S_0 之后，开始 S_1, S_2, \dots, S_n 个语句的并行执行，只有在它们全部完成以后，才开始执行 S_{n+1} 语句。并行语句也可以嵌套。例如：

```
begin  
  S0;  
  Cobegin  
    S1;  
    begin  
      S2;  
      Cobegin S3;S4;S5 Coend  
      S6;  
    end  
  S7;  
  Coend  
  S8;  
end
```

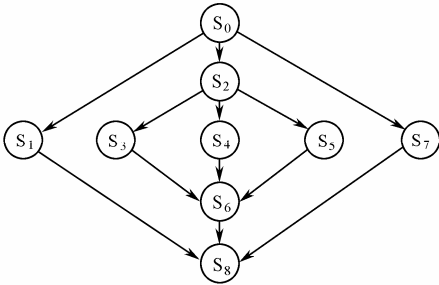


图 6-28 嵌套并发进程

该程序执行过程如图 6-28 所示。并行语言还需考虑程序分支、循环、并发进程间通信与同步，数组和进程组处理等。

2. 并行编译

要编出算术表达式的并程序可以依靠程序员掌握的并行算法，或依靠并行编译程序。有一些编译算法，可以经过或经过逆波兰表达式，直接从算术表达式产生能并行执行的目标程序。例如，有下列表达式

$$Z = E + A * B * C / D + F$$

利用普通串行编译算法，产生的三元指令组为

- 1 *AB
- 2 *1C
- 3 /2D
- 4 +3E
- 5 +4F
- 6 =5Z

指令间均相关，需 5 级运算。如采用并行编译算法可得

- 1 *AB
- 2 /CD

3 *12
4 +EF
5 +34
6 =5Z

其中，1，2 为第 1 级；3，4 为第 2 级；5，6 为第 3 级。

它可以分配给两个处理机，只需三级运算。可见有了好的并行编译算法，算术表达式也需要变形。

3. 描述程序并行性的指令

并程序的执行是一个不断进行并行性任务的派生和汇合的过程。派生是指在一个任务执行的同时，派生出可与它并行的一个或多个任务，分配给不同的处理机完成。这些任务可以是互不相同的，执行时间也不一样，要等它们全部完成以后再汇合起来，进入后继任务。后继任务可以是单任务，或新的并行任务。若是并行任务，则又开始派生和汇合过程。依次类推，直至整个程序结束。

在机器语言中，描述派生和汇合关系的并行控制指令通常用 **FORK**（派生）和 **JOIN**（汇合）指令。

指令格式：**FORK A**

功能：

① 在遇到 **FORK** 指令时，执行该指令的原进程，并根据标记符 **A** 派生出该标记符所对应的新进程，即准备好启动新进程或恢复原来进程继续执行时的现场。若共享主存，则应该产生存储器指针、映像函数、访问权等信息。

② 执行 **FORK** 指令的原进程，继续在分配给它的处理机上运行。

③ 将空闲处理机分配给被 **FORK** 指令派生的新进程，如果所有处理机均忙，则让新进程进入队列排队等待。

指令格式：**JOIN N**

功能：

① **JOIN** 指令有一个计数器，初始值置为 0，当执行 **JOIN** 指令时，计数器加 1，并与 **N** 比较。

② 若计数器值等于 **N**，说明它是执行中的第 **N** 个进程经过 **JOIN** 指令，则允许该进程通过 **JOIN** 指令，在其所在处理机上继续执行后继指令。

③ 若计数器值小于 **N**，则必须等待 **N** 个并行任务中尚未执行或虽然执行但未结束的进程到达 **JOIN** 指令。现在执行 **JOIN** 指令的这个进程可以先结束，并把占用的处理机释放出来，分配给排队等待的其他任务。

现以并行编译中的例子 $Z=E+A*B*C/D+F$ 说明如下：

S₁ G=A*B
S₂ H=C/D
S₃ I=G*H
S₄ J=E+F
S₅ Z=I+J

如果不加并行控制指令，该程序仍是串行程序，不能发挥多处理机作用。图 6-29（a）

表示了指令间数据相关情况。采用 FORK 和 JOIN 指令所反映的并行关系如下：

```

      FORK S2
S1  G=A*B
      JOIN 2
      GOTO S3
S2  H=C/D
      JOIN 2
S3  FORK S4
      I=G*H
      JOIN 2
      GOTO S5
S4  J=E+F
      JOIN 2
S5  Z=I+J

```

它的执行过程如图 6-29 (b) 所示，它需要两个处理机 PU₁ 和 PU₂。

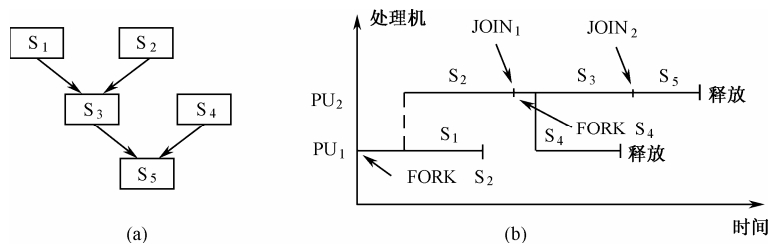


图 6-29 数据相关和并行执行

6.3.4 多处理机的操作系统

多处理机的操作系统在许多方面与分时操作系统具有相似的功能。其相同功能是：资源分配和管理、存储器和数据保护、防止系统死锁、异常进程的终结和处理等。此外，多处理机系统还需要具备下列功能：高效率地利用资源，合理地进程调度，使输入、输出和处理机负载平衡；在出现故障时，使系统重新配置，适度降级运行，以提高系统的可靠性。所有这些要求和多处理机的运行环境给操作系统增加了很多负担，要求它自动地发掘硬件和运行中程序的并行性。如果操作系统的性能达不到一定的水平，则多重处理的特长也就无从发挥了。因此，高效的多处理机操作系统是多处理机系统软件的核心。处理机数目大量增加以及处理机模块化和处理机之间的互连网络均会影响系统开发。此外，通信方式、同步机构、布局和分配策略对操作系统的性能起决定性作用。

1. 多处理机操作系统的分类

多处理机操作系统按其结构可分为：① 主从结构 (Master Slave Configuration)，即专用控制方式；② 单独管理 (Separate Supervisor)，即分布控制方式；③ 浮动管理控制 (Floating Supervisor Control)，即对称控制方式。目前多数多处理机操作系统采用主从方式，因为它简单，容易实现，把单处理机的分时操作系统进行适当扩充，就可设计出主从方式的操作系统。但是，它在控制使用系统资源方面效率不高，而后两种方式均比主从方式好。

（1）主从方式操作系统

① 由一台主处理机进行系统的集中控制。主处理机管理系统中所有处理机的状态，并对所有从处理机分配任务。操作系统只运行在主处理机上，它把从处理机视作可调度的资源。

② 从处理机经过自陷（Trap）或管理调用指令向主处理机发出请求，主处理机中断当前程序，识别该请求并完成相应的服务。由于只有主处理机执行管理程序，所以它不需要也不必考虑重入问题，避免了系统控制表格的冲突和封锁等问题。

③ 当主处理机出现故障时，整个系统崩溃，需要操作人员干预或重新启动。

④ 系统的硬件和软件比较简单，不够灵活。

⑤ 如果主处理机不能很快地为从处理机分配任务，从处理机可能长时间空闲，系统利用率会下降。

⑥ 本方式对工作负载固定，且从处理机能力比主处理机低的系统是适用的。例如，异构型多处理机系统，用本方式的操作系统比较有效。

（2）单独管理方式操作系统

① 每个处理机均有一个独立的管理程序（操作系统的内核）在运行，即每一个处理机都有同一个内核的副本为其本身服务。

② 由于处理机之间可交互作用，因此管理程序的某些代码必须是可重入的，或为了给其他处理机提供副本而必须重复设置。

③ 每个管理程序都有一套自用表格，但仍有一些共享表格，从而带来共享表格访问冲突的问题，使进程调度的复杂性和开销增大。因此实现比较困难。

④ 适应分布处理模块化结构的特点，减少了对控制专用处理机的需求，有较高的可靠性和系统利用率。

⑤ 每个处理机有自己的 I/O 设备和文件，因此整个系统的 I/O 结构有变动时，需要人工干预。

⑥ 由几台处理机共同负担整个系统的控制，所以当出现故障时，自动启动一台出故障的处理机相当困难，需要人工干预。

（3）浮动管理控制方式操作系统

① 它属于上述两种方式的折中。“主处理机”可以从一台处理机向另一台处理机浮动，主控制程序也可以转移，或者几个处理机同时执行管理程序。担任主处理机时间也不固定。

② 该方式可使各类资源的工作负载比较平衡。

③ 通过静态设置或动态控制的优先级，安排服务请求的次序。

④ 由于若干处理机可以同时执行同一个服务程序，因此管理程序的大多数代码必须是可重入的。

⑤ 由于存在多个管理程序，所以表格访问冲突和表格封锁延迟是不可避免的，因此必须保护系统的完整性。

⑥ 该方式在硬件和可靠性上具有单独管理方式的优点，而在操作系统的复杂性和经济性上则接近于主从方式。

必须指出，实际的多处理机系统，其操作系统很少严格地归属于某一种方式，往往是混合式的，且有发展和变化，所以必须具体系统具体分析。

2. 资源分配和进程调度

资源分配和进程调度是决定多处理机系统效率的关键。对多处理机而言，资源分配的核心问题是处理机分配。按进程提出的要求分配处理机是进程调度的主要内容。对于异构型多处理机，根据功能专用化和功能分布原则，把控制、I/O、处理（语言翻译和解释、机器仿真、应用程序运行、数据库管理等）分配给各个专门处理机完成。为了保持处理机负载平衡，在信息流量变化的情况下，要进行动态分配。对于同构型多处理机，根据资源池原则，将处理机、内存、I/O 通道等各种资源放在一个公共的资源池里，由多个进程共享，达到设备的高使用率和处理机的负载平衡。

进程控制主要指进程切换。进程切换是指任何进程由于某种原因（如等待外部条件满足、转 I/O、转其他专用处理机等）而不能继续进行时，要求将处理机释放，分配给其他等待的进程使用，从而改变进程状态。进程状态有 4 种：运行（执行）、挂起（封锁）、等待和就绪。运行状态是指某进程得到处理机和其他资源正在执行指令。挂起状态是指某进程把它占用的处理机和内存及其他资源释放出来，程序执行处于暂停。在运行和挂起之间有一个中间过渡状态，以便进行一个“测试”操作，避免传送的信息进入错误的地址，该状态称为等待状态。已经挂起的进程，遇到外部条件满足，在重返运行状态之前先经过一个中间过渡状态，以便排队等待分配处理机，该状态称为就绪状态。4 种状态切换如图 6-30 所示。

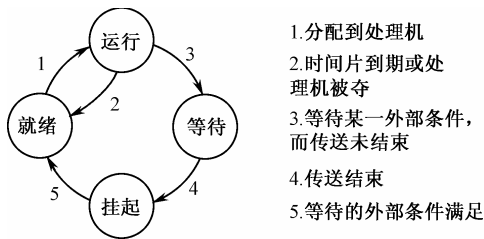


图 6-30 4 种进程状态的切换

进程状态的切换需要信号控制，表现为有关的操作系统原语，它是程序调度的主要组成部分。当进程需要从运行状态转向挂起状态时，由其本身或其他进程向调度程序发出挂起请求，在操作原语 **Suspend-Process** 的控制下，执行 **Test-Event** 原语。如果测试出有信息正在传送，则过渡到等待状态。执行 **Test-Event** 测试一个公共变量，但不改变它的值。该公共变量用于信息传递互斥的信号灯，只能由 **Send** 原语减值和 **Receive** 原语增值，其原理就像 **P** 和 **V** 操作一样。信息传递结束，信号灯增值，允许进程从等待转向挂起。这时，要保留现场和释放资源，即把原进程的状态向量（包括程序状态字、通用寄存器内容、地址空间和内容、I/O 设备状态等）保存起来，将释放的处理机转给就绪队列中的新进程，取出新进程的状态向量，继续进行。若找不到新进程，则该处理机暂时进入空闲状态。当某进程被挂起后，它自己不能发出“苏醒”信号。由其他进程发出信号，在 **Resum-Process** 原语的控制下将挂起的进程转入就绪状态，进入队列等待分配处理机。因此进程进入挂起状态前，应预做安排。例如，依靠另一个进程对某一信号灯增值，或某个 I/O 服务程序结束时发出信号。

在进程挂起前，要求释放所有占用的资源，这是防止死锁的一种办法，但增加了状态切换的开销。如果允许挂起的进程保留除处理机之外的其他资源，则要采取有效措施防止死锁。造成死锁有下列 4 个必要条件：

- （1）进程排他性地占有某些系统资源；
- （2）当进程对资源进一步要求被拒绝而挂起时，已占用资源仍不释放；
- （3）不能预先分配资源；

(4) 资源占用状况出现死循环，即 A_1 要求的资源被 A_2 占用， A_2 要求的资源又被 A_3 占用，……依次类推，最后， A_n 要求的资源又被 A_1 占用。

- 防止死锁，可以从上述任一个环节打破，即上述至少有一个条件不能满足。例如，
- (1) 进程挂起后，强制放弃已占用资源，等苏醒后再重新申请。
 - (2) 进程必须对所需全部资源一次性提出申请，在未满足时不占有任何资源，一旦满足，在整个运行期间抓住不放。
 - (3) 在进程需要有多种资源的情况下，对资源规定一个固定次序，各进程按此次序先后提出申请，以避免几个进程同时要求某资源而出现死循环。

(4) 把系统可用资源数、各进程一次需用的最大资源数，以及每个时刻实际分配的资源数列成表。在进程提出资源申请时，由操作系统进行核算，若满足它的需求（已登记在表内）后，看是否会造成系统死锁。

防止死锁很重要，但这不是多处理机系统资源管理的全部，还需从保证系统的高效率出发，解决进程调度和资源分配的问题。所以在多道程序中，行之有效的方法并不完全适用于以任务级并行为特征的多处理机系统。因为任务之间有复杂的内在联系，调度策略不能简单地用优先级、时限等准则来表示。所以合理、快速、高效的进程调度算法，以及资源全方位合理配置和调度是多处理机操作系统需要重点解决的问题。只有这些问题与并行算法一同解决，多处理机系统才能发挥其巨大的潜在能力。

6.4 多处理机系统实例

6.4.1 C_m^* 多处理机

C_m^* 多处理机由美国 Carnegie Mellon 大学研制，共包括 50 台 DEC LSI-11 微机，用三级总线连成一个松散耦合的多处理机系统，其结构如图 6-31 所示。最基层一级 LSI-11 微机，

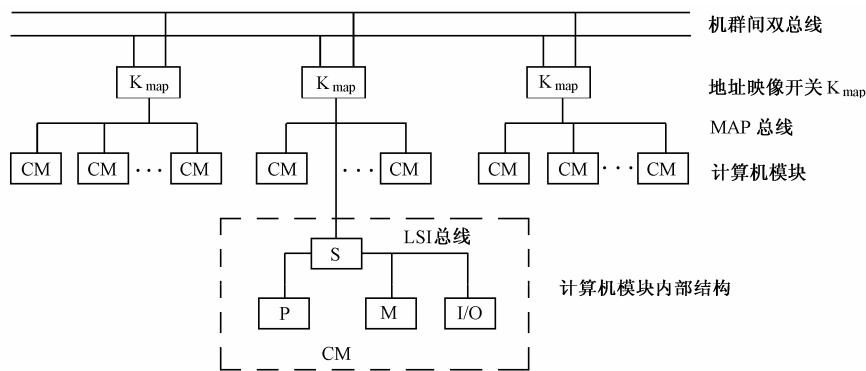


图 6-31 C_m^* 多处理机结构

内有 LSI 总线和开关 S ， S 的作用是使处理机 P 与本地存储器 M 和本地 I/O 通信，或者通向第 2 级总线 MAP。MAP 总线每个机群一套，把机群内部的计算机模块 CM（即 LSI-11 微机）连接起来，最多可达 14 个。每个机群有一个地址映像开关 K_{map} ，为了加快总线通信频宽，机群间为双总线，各机群 K_{map} 挂上双总线。由于 C_m^* 是分布处理系统，每个 CM 内有 28KB

本地存储器，各个 CM 的存储器组织在一起，构成了有 28 位地址的虚拟存储空间 ($2^{28}=256\text{MB}$)。每个处理机用虚拟地址访问存储器时，由 K_{map} 进行地址变换，以确定访问本地存储器或机群内不同的存储器。因此，关键的操作系统原语移入 K_{map} 与其他 K_{map} ，把 CM 从管理功能中解脱出来。 K_{map} 结构如图 6-32 所示。一个 K_{map} 由数据存储器 and 三个处理部件组成。其中， K_{bus} 是总线控制器，它仲裁 MAP 总线请求；Linc 用于管理本地 K_{map} 和其他 K_{map} 间通信； P_{map} 是映像处理部件，它响应 K_{map} 和 Linc 的请求，完成“虚-实”地址变换。三个处理部件通过三个队列接口。由于 K_{map} 比 CM 中存储器快得多，因此它能按多道程序处理多达 8 个并发请求。含有虚拟地址的请求加入到运行队列，经 P_{map} 转换成物理地址，加入到输出队列，由 K_{map} 按此物理地址启动目的 CM 的存储器，与此同时， P_{map} 可对运行队列中的下一个请求进行虚-实地址转换。

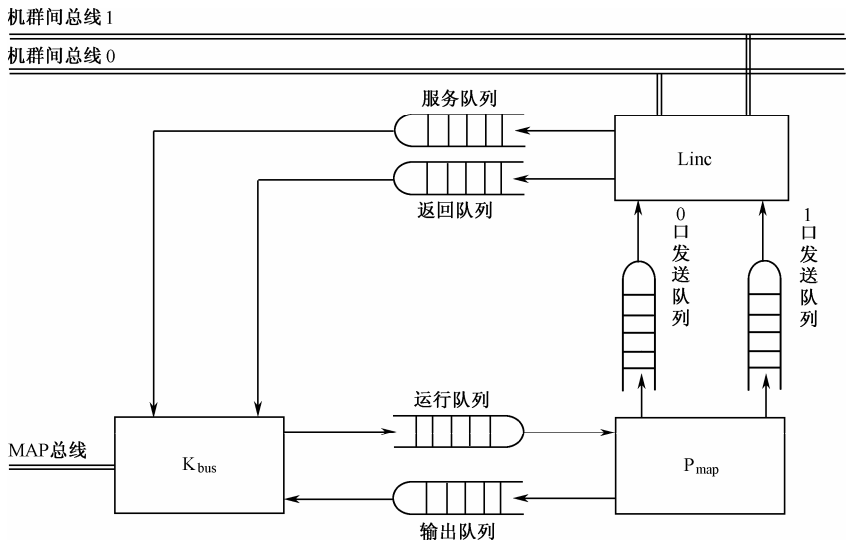


图 6-32 K_{map} 结构

如图 6-33 所示，在机群内部进行存储器访问，其访问过程如下：

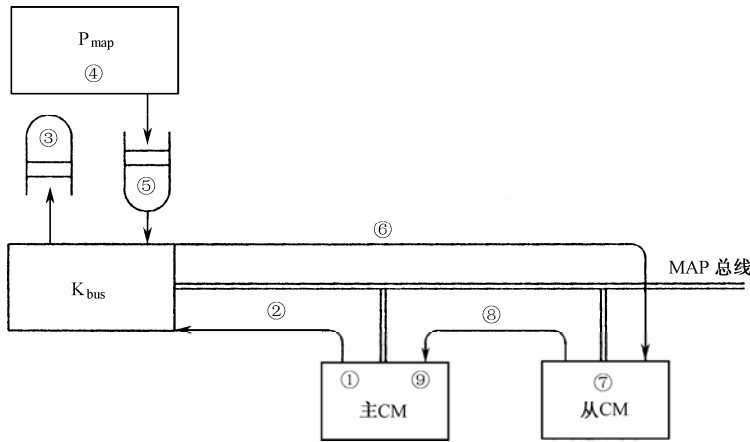


图 6-33 机群内部存储器访问过程

- ① 某 CM（称为主 CM）启动一次非本地存储器的访问；
- ② K_{bus} 从主 CM 读虚拟地址；
- ③ 该虚拟地址的请求进入运行队列等待；
- ④ P_{map} 完成“虚-实”地址转换；
- ⑤ 含物理地址的请求在输出队列中等待；
- ⑥ K_{bus} 发送物理地址到从 CM（目的 CM）；
- ⑦ 从 CM 从它的处理机中窃取一个存储周期，进行存取操作；
- ⑧ K_{bus} 控制结果返回主 CM；
- ⑨ 处理机继续进行后继的工作。

不同处理机 CM 之间经机群间总线通信，机群间存储器访问如图 6-34 所示，其访问过程如下：

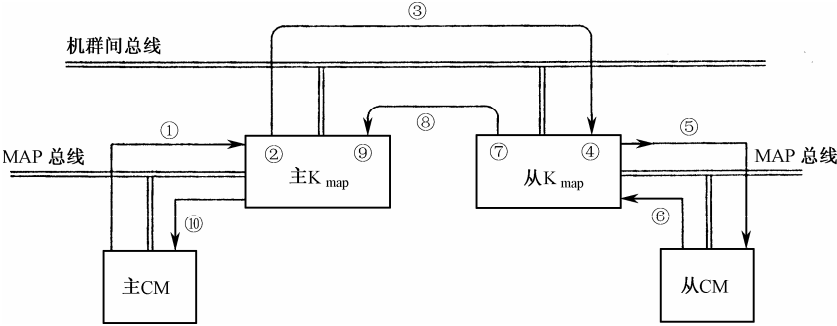


图 6-34 机群间存储器访问过程

- ① 主 K_{map} 从主 CM 接收请求；
- ② 主 K_{map} 准备一个机群间报文；
- ③ 该报文传送到从 K_{map} ；
- ④ 从 K_{map} 对该报文进行译码；
- ⑤ 该请求发送到从 CM，进行存储器访问；
- ⑥ 从 CM 将结果返送到从 K_{map} ；
- ⑦ 从 K_{map} 准备一个返回的机群间报文；
- ⑧ 该报文返回到主 K_{map} ；
- ⑨ 主 K_{map} 接收返送的报文；
- ⑩ 结果送到主 CM。

C^*_m 的 MAP 总线和机群间总线都采用报文分组交换方式，因此本地通信需 $3.5\mu s$ ，机群内通信需 $9.3\mu s$ ，机群间通信需 $26\mu s$ ，体现了松散耦合的特点。根据程序的局部性原则，本地访问的命中率可达 90% 以上，从而使平均通信时间降到 $4.1\mu s$ 。又由于采用分布式 K_{map} 开关结构，当一个机群发生故障时，整个系统仍能在降级方式上操作，体现了分布处理系统的优点。

6.4.2 C_{mmp} 多处理机

C_{mmp} 多处理机也是美国 Carnegin Mellon 大学研制的。由 16 台 DEC PDP-11/40E 经一个 16×16 交叉开关连到 16 个存储器模块上，组成一个紧密耦合的多处理器系统，其结构如

图 6-35 所示。图中也表示了单个处理机模块的结构。每个处理机模块有一个 8KB 的本地存储器 LM，用于存放操作系统。I/O 设备只连到指定的处理机的单总线上。因此，一个处理机模块不能直接启动不在其单总线上的 I/O 设备，必须经过操作系统完成，它对用户是透明的。每个处理机模块有一个与处理机间总线相连的接口 K_{ibi} ，用于定义处理机在处理机间总线上的地址。处理机间总线用于各模块 CM 间的通信，有一个公共的时钟 K_{clock} 和控制 $K_{interbus}$ 。

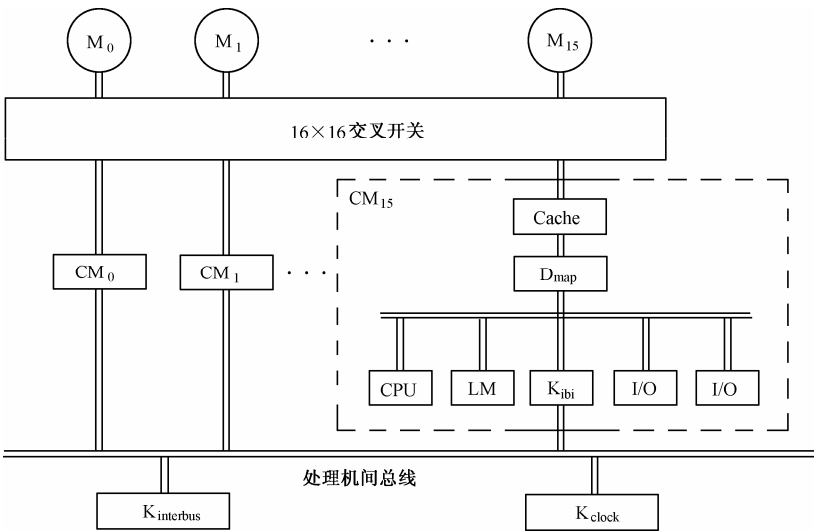


图 6-35 C_{mmp} 多处理机结构

用小型机构成大型计算机系统的最大限制是它的地址空间小。在 C_{mmp} 中，处理机地址为 16 位，单总线地址为 18 位，共享存储器地址为 25 位。为此，每个处理机模块都有地址定位部件 D_{map} ，实现存储器地址变换。处理机模块产生的地址分为 8 页，每页 8KB，单总线地址分为 32 页（因为单总线地址增加了 2 位），共享存储器分为 4096 页。 D_{map} 中地址变换机构如图 6-36 所示。单总线地址增加的 2 位是从程序状态字 $PSW<7:8>$ ，即由第 7 位和第 8 位得到的。这两位定义 4 个公共堆栈页，每页有 8 个映像寄存器。由 16 位处理机地址中的 $<13:15>$ 高 3 位选择其中之一，该寄存器提供 12 位的页地址，处理机地址的其余 13 位为页内地址，两个地址拼接构成 25 位存储器地址。

在映像存储器中，除了 12 位用于页地址外，其余 4 位 $<12:15>$ 用于控制，表示该页的性质和状态，如页装入位、已修改位、可进入 Cache 位等。25 位的存储器地址，经 16x16 交叉开关传输，其高 4 位 $<21:24>$ 用于指定相应存储器模块，而 $<13:20>$ 位选择该模块内某页， $<0:12>$ 位为该页内某单元地址。如果所有处理机选择 16 个不同存储器模块，则交叉开关可以得到最大的并行性。从地址变换交叉经开关和传输，总开销为 $1\mu s$ 左右，存储器访问时间也近于 $1\mu s$ 。所以 C_{mmp} 是紧密耦合的多处理机系统。

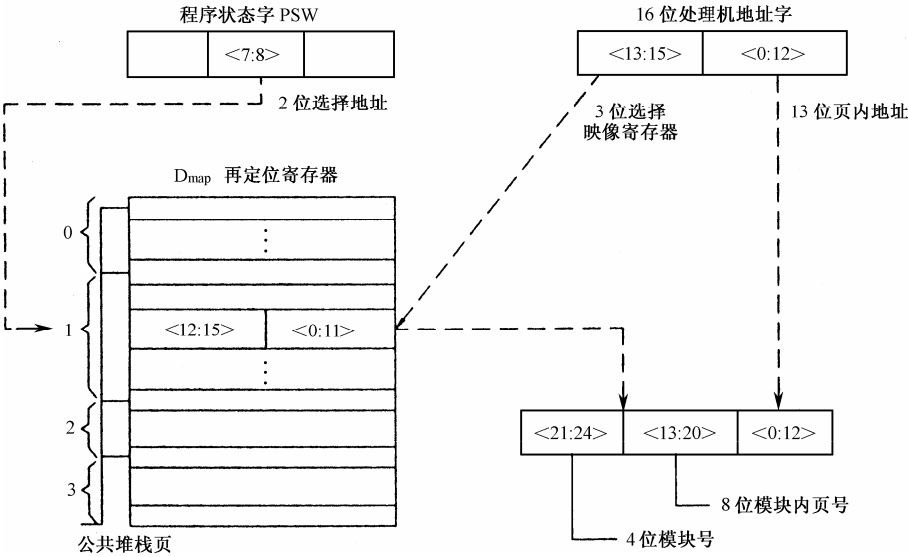


图 6-36 D_{map} 中的地址变换机构

第 7 章 RISC 结构

精简指令集计算机（Reduced Instruction Set Computer, RISC）的产生是相对于传统的复杂指令集计算机（Complex Instruction Set Computer, CISC）结构而言的。尽管 1979 年美国加州大学伯克利分校的帕特逊等人就提出了这个名词，但不同看法颇多，至今尚未有对 RISC 的严格定义。人们普遍认为，RISC 应该是一种计算机设计的基本原则，它的出现是计算机系统结构发展史上的一个重要里程碑。

7.1 RISC 结构概述

7.1.1 传统计算机系统结构的设计思想

传统的计算机系统结构有过几次重大的发展，都基本遵循了冯·诺依曼结构的原则。20 世纪 60 年代，IBM System 360 的出现第一次明确了计算机系统结构是机器程序设计员为编写程序所看到的一个计算机的抽象结构，而计算机组织是实现这个结构的硬件组成。由此产生了一个重要概念，即对于某一个产品系列，其计算机系统结构可以是相同的，系列中每一个档次产品的计算机组织可以有差别，但各个档次的产品在软件上具有兼容性。这样，可以针对不同要求的用户，提供不同层次、不同价格的计算机产品，而且用户使用的软件可以向上兼容，从而有效地保护用户的投资。

微程序设计是实现产品系列化的一项重要的系统结构技术。由于计算机指令系统的性能不断增强，就要求扩充微程序的存储器，这也使传统的计算机系统结构的设计思想更为突出地表现在以下三个方面：① 指令系统越丰富越好；② 指令系统功能越强越能改善系统结构的质量；③ 指令系统越复杂越便于软件的兼容。由此造成的后果是微程序的微代码量越来越大，微程序的存储器容量越来越大，甚至要有专门的编写微程序的程序设计语言。

7.1.2 RISC 设计思想的产生

1. 20%-80%定律

传统的 CISC 计算机指令系统随着计算机的发展引入了各种各样的复杂指令，使得指令系统和为实现这些指令系统而设计的计算机系统结构越来越复杂。经过大量的研究和分析发现，在 CISC 指令系统中，各个指令的使用频度相差悬殊，如表 7-1 所示。大概有 20% 的指令反复被使用，使用量占整个程序的 80%；而有 80% 左右的指令很少被使用，其使用量占整个程序的 20%。这就是所谓的 20%-80% 定律。

分析过程中还发现，这些很少使用的指令，相对经常使用的指令来说更复杂。计算机为了实现这些很少使用的指令，增加了计算机系统结构的复杂性。

表 7-1 指令系统运行的数据统计与分析

按使用频度排序			按执行时间排序		
指 令	占百分比	累计百分比	指 令	占百分比	累计百分比
1. MOV	24.85	24.85	1. IMUL	19.55	19.55
2. PUSH	10.36	35.21	2. MOV	17.44	36.99
3. CMP	10.28	45.49	3. PUSH	11.11	48.10
4. JMPcc	9.03	54.52	4. JMPcc	10.55	58.65
5. ADD	6.80	61.32	5. CMP	7.80	66.45
6. POP	4.14	65.46	6. CALL	7.27	73.72
7. RET	3.92	69.38	7. RET	4.85	78.57
8. CALL	3.89	73.27	8. ADD	3.27	81.84
9. JUMP	2.70	75.97	9. JMP	3.26	85.10
10. SUB	2.43	78.40	10. LES	2.83	87.93
11. INC	2.37	80.77	11. POP	2.61	90.54
12. LES	1.98	82.75	12. DEC	1.49	92.03
13. REPN	1.92	84.67	13. SUB	1.18	93.21
14. IMUL	1.69	86.36	14. XOR	1.04	94.25
15. DEC	1.37	87.73	15. INC	0.99	95.24
16. XOP	1.13	88.86	16. LOOPcc	0.64	95.88
17. REPNZ	0.78	89.64	17. LDS	0.64	96.52
18. CLD	0.54	90.18	18. CMPS	0.44	96.96
19. LOOPcc	0.52	90.70	19. MOVS	0.39	97.35
20. TEST	0.40	91.10	20. JCXZ	0.37	97.72
21. SYI	0.40	91.50	21. LODS	0.31	98.03
22. LDS	0.39	91.89	22. REPN	0.28	98.31
23. LODS	0.35	92.24	23. INTcc	0.26	98.57
24. AND	0.35	92.59	24. STOS	0.25	98.82
25. SHR	0.33	92.92	25. SHR	0.23	99.05
26. STOS	0.31	93.23	26. LEA	0.12	99.17
27. MOVS	0.29	93.81	27. OR	0.11	99.28
28. JCXZ	0.29	93.81	28. REPNZ	0.11	99.39
29. SH/AL	0.27	94.08	29. AND	0.09	99.48
30. CMPS	0.27	94.35	30. TEST	0.09	99.57

2. 软、硬件设计的折中

随着 VLSI 技术的迅速发展，研究人员对系统设计中硬件与软件的复杂性应该如何优化进行了深入的研究。他们普遍认为，要使一个系统具有较高的性能价格比，单靠增加硬件的复杂性是不行的，必须把硬件和软件结合起来相互配合，均衡考虑，才能提高性能价格比。

3. VLSI技术的发展

VLSI 技术突飞猛进地发展，使得在一块芯片上能方便地做出大量的寄存器，促使系统设计者能使用较快的“寄存器-寄存器”指令。这样指令系统就可以更加精简，控制部件更加

简化, 整个系统效率更高。

7.1.3 RISC系统结构的特点

虽然 RISC 结构在本质上还有冯·诺依曼结构的许多特征, 但其自身在系统结构和实现技术上有许多不同的特点。

1. RISC结构的设计原则

要使计算机系统设计达到最高的有效速度, 需将那些对系统产生净增益的功能用硬件来实现。如大量增加内部通用寄存器堆, 可以使操作仅在内部寄存器间进行, 从而提高了速度。其他系统功能则可以用软件来实现。

RISC 结构的另一个设计原则是, 排除实现复杂功能的指令, 仅保留确能提高机器性能的指令, 以达到使指令集精简的目的。为此采用了一些方法: 选用那些使用频度最高的指令, 补充经分析最有用而实现又不复杂的指令; 使每一指令都在一个机器周期内完成, 提高处理速度; 使指令集内指令长度是固定的, 便于时序节拍的最佳安排; 只有几条存数、取数的访内指令, 大部分指令操作都在内部寄存器之间进行, ……这些原则使得在指令功能设计、指令格式设计、指令编码设计上都有了许多自己的特色。

RISC 结构的设计要求尽可能地简单高效, 这就必须考虑对高级语言的支持程度。因此, RISC 结构的设计原则中又将编译器作为机器的基本功能。由于指令集的精简, 使编译工作得到简化; 大量操作在内部寄存器间进行, 不再需要大量的编译工作来寻址操作; 一个机器周期完成一条指令操作, 编译器易于调整指令流, ……

2. RISC技术的特点

(1) 指令功能与指令执行周期数的权衡。CISC 设计中往往追求指令功能的复杂, 甚至希望一条指令的功能相当于一条高级语言的语句。指令功能复杂使得硬件实现的复杂程度大大提高, 也使指令的周期延长或增加, 这样反而降低了计算机的性能。

RISC 设计中希望避免这种情况, 需要在指令功能复杂程度与指令执行周期数之间有一个很好的权衡。应该优先考虑那些经常使用的基本指令的实现, 对于那些功能复杂、硬件实现也复杂的指令是否引入 RISC 的指令集要采取慎重的态度。

(2) 引入多级指令 Cache。RISC 结构仅有存数、取数指令才访问主存, 通过 Cache 与处理器中的寄存器堆进行寄存器与寄存器之间的高速运算。但采用 Cache 后就存在如何保证一条送数指令返回的结果与最近的取数指令所给出的相同地址结果的一致性问题。解决这个问题的一个简单想法是增加 Cache 的命中率, 而提高命中率就要增大 Cache 的容量, 考虑到高速 Cache 的成本, 往往采用多级 Cache 结构来实现。当然 Cache 的一致性问题还涉及 Cache 的组织结构、替换算法和写控制策略等。在多处理器系统中, 还要同时更新所有处理器的 Cache 内容, 以保证 Cache 的一致性。这也涉及进程的动态调度、I/O 共享系统总线等复杂关系。

(3) 面向存储器堆的结构。过去的访内指令功能看起来很强, 其实执行效率很低。原因在于 CPU 与存储器数据的传输不仅仅是芯片与芯片之间的传输, 有可能是 CPU 板与存储器板之间的传输。它们的传输频宽远不能与 CPU 和寄存器堆的芯片内传送频宽相比。

RISC 结构设计就是面向寄存器堆结构, 重视“寄存器-寄存器”操作指令, 充分利用 VLSI 技术中的高速芯片上的频宽来进行数据传输。同时, 大量使用“寄存器-寄存器”指令, 使指令控制逻辑简化, 缩小占用的芯片面积, 为在一片芯片上增加更多的寄存器堆提供了可能。

（4）充分提高流水线的效率。流水线技术已在一般计算机的设计中被广泛采用，在 RISC 技术中更为重视。即使 RISC 结构采用了流水线结构的并行技术，要使一条指令在一个机器周期内完成，甚至一个机器周期内完成几条指令的想法仍是 RISC 结构设计者梦寐以求的。因此，产生了单发射结构（即在一个机器周期内发射一条指令）和多发射结构（即在一个周期内发射多条指令）。也出现了一些属于指令级并行处理的新结构，以达到在一个机器周期内执行多条指令的目的。

① 超级流水线方式。将原来的流水线进一步细化，达到缩短各级执行时间的目的。如一般 RISC 结构的流水线是 4~5 级，而超级流水线则是 8 级以上。

② 超标量方式。在机器里设置多条流水线，同时执行多个处理。找出能够并行译码的指令，并在指令执行阶段动态执行调度。它往往支持 32 位的指令格式，便于目的代码的生成。

③ 超长指令字（VLIW）方式。它与超标量方式类似，也由多条流水线并行处理多条指令。但它的不同在于，指令格式往往大于 32 位，它是在编译阶段进行程序调度的。因此，一条指令可以指定多个处理，具有指定处理的多个字段。

当然，流水线也会“阻塞”，造成“断流”。影响流水线效率的主要因素是数据相关和转移相关。

（5）指令格式的简单化和规整化。RISC 结构的指令基本上是一个字（32 位）长度，而且指令中操作码字段、操作数字段都尽可能具有统一的格式。格式的规整也使指令的操作规整，这样有利于流水线的执行，提高译码操作的效率，并使译码控制逻辑简化。

（6）RISC 技术中的编译技术不仅要生成代码，而且要优化代码。RISC 技术强调编译优化技术，编译出来的代码要重新组织、调度指令的执行次序，充分利用 RISC 的内部资源，发挥其内部操作的并行性，从而提高流水线的执行效率。例如，编译优化时，将参加操作的操作数尽可能地放在寄存器堆内反复使用，减少访问存储器的操作次数。

RISC 结构的指令系统简化，使编译时间延长。用编译时间换取运行的高效率，是值得的。因为，程序编译一次，其生成的优化执行代码可以高效率地执行多次。在 RISC 设计时要充分考虑硬件复杂性和软件复杂性之间的权衡。

7.1.4 RISC 的定义

由于 RISC 是一种设计思想，它还在不断地发展和丰富，因此要准确地给 RISC 下定义是困难的。有两种说法可以比较明确地说明 RISC 的设计思想。

1. 美国卡内基-梅隆（Carnegin Mellon）大学的定义

（1）指令系统的大多数指令只需执行简单和基本的功能，其执行过程是在单个机器周期内完成的。

（2）由于存储访问指令执行时间长，应尽量减少这类指令。采用存取指令结构（LOAD STORE 结构），即只保留 LOAD 指令和 STORE 指令。面向运算部件的操作数都经过 LOAD 指令和 STORE 指令，使数据从内存预先放在寄存器堆内，加快执行速度。

（3）芯片逻辑不采用或少采用微码技术，而采用硬联逻辑，减少指令解释的开销。

（4）减少指令数和寻址方式，使控制部件简化，加快执行速度。

（5）指令格式固定，指令译码简化。

（6）编译开销很大，应尽可能优化。

2. IEEE的迈克尔·斯莱特 (Michael Slater) 的定义

RISC 处理器应具有使流水线处理器能有效地执行、使优化编译器能生成优化代码而设计的指令集。

(1) RISC 结构为使流水线有效地运行, 应具有以下特征: ① 简单而且统一格式的指令译码; ② 大部分指令可以单周期执行; ③ 只有 LOAD-STORE 指令访问存储器; ④ 简单的寻址方式; ⑤ 延迟转移; ⑥ LOAD 延迟。

(2) RISC 结构为使优化编译器便于生成优化代码, 应具有以下特征: ① 三地址指令格式; ② 较多的寄存器; ③ 对称的指令格式。

7.1.5 关于CPI的讨论

CPI (Cycle Per Instruction) 是衡量一条指令执行的平均周期数。RISC 结构的设计思想就是要使 CPI 进一步减小, 即 $CPI \leq 1$ 。它的一般公式为

$$P = ICT$$

式中, P 是执行一个程序所花费的时间; I 是这个程序所需执行的总指令数; C 是每条指令执行的平均周期数, 即 CPI; T 是周期。

对 RISC 和 CISC 的 I , C , T 三项值的比较如下。

(1) I 值。对同一个程序编译后生成的执行代码, RISC 的 I 值要比 CISC 的大。因为 CISC 的一条复杂指令可以抵好多条 RISC 的精简指令, 但是根据 20%-80% 定律简单指令使用得多, RISC 的 I 值是 CISC 的 1.3~1.4 倍。

(2) C 值。CISC 的指令实现往往是几个周期完成的, 因此 CISC 的 C 值是 4~6 甚至更高。而 RISC 的指令基本上是在单个周期完成的, 即使把访内指令也计入, C 值也只是 1.1~1.4。

(3) T 值。机器周期与主频有关, 一般 RISC 结构简单, 机器周期比 CISC 结构的机器周期略短。

由此可见, RISC 结构的 P 值远比 CISC 结构的小, 这也是目前计算机系统设计普遍采用 RISC 的原因所在。

现在的计算机设计者追求的是如何在 RISC 结构上不断地减小 I , C , T 值。高效率的编译技术和改进算法可以减小 I 值; 提高处理器的主频和改善系统结构可以减小 T 值; 而减小 C 值则是 RISC 结构的主要方面, 就是要进一步提高 RISC 结构的内部并行性。有些措施对 RISC 结构的发展是至关重要的。如 Cache 分为指令 Cache 和数据 Cache; 用提高 Cache 的容量来提高命中率; 设立多端口的寄存器堆; 超标量结构、超流水线结构等。

7.2 流水线结构

如果流水线正常执行而无阻塞, 理论上就可以做到每个周期执行一条指令。但实际上, 流水线的执行效率因受到数据相关和转移相关的影响而降低。

图 7-1 表示一个 4 级功能段的流水线结构。指令 n 与指令 $n+1$ 是在流水“管道”中正常运行的。但假如 $n+2$ 指令的操作数要用到指令 $n+1$ 的结果, 而指令 $n+1$ 的 WR 流水级尚未把结果写回寄存器堆, 则第 $n+2$ 条指令的执行级 EX 就无法正常执行。它必须等待, 等到 WR

完成后再取到操作数执行操作。这个等待的周期就叫“气泡”，流水线受阻，效率就会降低。

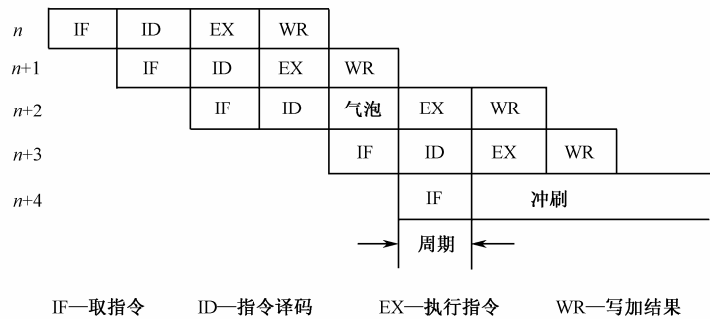


图 7-1 一个 4 级功能段的流水线执行和相关性问题

第 $n+1$ 条指令与第 $n+2$ 条指令之间产生的是数据相关。数据相关可以用硬件检测和内部推前的方法解决，或依靠软件上的指令调度方法解决。

假如第 $n+3$ 条指令是一条转移指令 JUMP，则第 $n+4$ 条指令应该执行转移后的目标地址指令。然而，等到在第 $n+3$ 条指令的译码流水级 ID 译出这条转移指令时，顺序排列的第 $n+4$ 指令已经执行取指令级 IF，即第 $n+4$ 条指令已经进入流水“管道”了，但实际上应该执行的却是转移后的目标地址指令，这就发生了差错。这种情况叫做转移相关。过去，原始的解决方法是“冲刷”掉流水管道中的第 $n+4$ 条指令，等到转移目标地址指令找到以后，再把它放入流水管道执行，但这又会降低流水线的执行效率。

转移指令有三种：无条件转移、有条件转移、循环转移。这三种在通常程序设计中表现出的转移，在程序整体指令中占据相当大的百分比。据统计分析，前两种转移指令占据程序指令总数的 30%。如再加上循环转移，转移相关对流水线执行效率的影响更为严重。而数据相关（包括寄存器数据相关和 LOAD 数据相关）和程序类型的关系很密切，较难得出统计平均数。但可以看出，这些相关性问题可使流水线的效率严重降低。编译优化及其指令调度的目的，就是通过重新组织指令执行的顺序，解除转移相关和数据相关或降低相关问题带来的损失程度。

单发射结构中流水线设计的目的是做到每个周期能平均执行一条指令，即 $IPC=1$ 。但由于转移相关和数据相关，以及其他的资源冲突，使 IPC 远小于 1。通过指令重组，可以提高 IPC 并使之接近于 1，但单发射的 RISC 的 IPC 不可能大于等于 1。

为了使 $IPC>1$ ，自然会联想到能否在一个周期内发出多条指令，这就是“多发射结构”。多发射结构常见的有超标量、超流水线和 VLIW 结构。此外，还有数据流结构也属于多发射结构这一范畴。超标量、超流水线和 VLIW 的流水线结构分别如图 7-2 (a)、(b) 和 (c) 所示。

图 7-2 所示的多发射结构的流水线是由 4 个流水级组成的，即 IF, ID, EX 和 WR。超标量 RISC 是在一个周期内同时发出三条指令，通常超标量 RISC 内具有多个执行部件，所以在执行流水级中，三条指令分发到三个独立的执行部件去分别执行。而超流水线结构是把每一个流水级（1 个周期）分成三个子流水级，而每一个子流水级中取出的仍只有一条指令，但总的来看，在 1 个周期内取出了三条指令。对于超流水线结构，其执行部件可以有一套或多套。不难看出，超标量结构的工作部件较多，即片上晶体管数目也需增多，但每个部件的工作速度相对要求可低一些。超流水线结构的工作部件较少，即片上晶体管数目增加不多，但每个部件必须在一个子周期内执行，所以要求的工作速度较高。概括地说，超标量结构是以

空间换取了时间，而超流水线结构是以时间换取了空间。当然，现在也有超流水线、超标量的 RISC 产品出现。DEC 公司的 Alpha 就是这种结构。Intel 公司的 Pentium 也属于超流水线、超标量结构。其他大部分新一代 RISC 都属超标量结构，如 i 860, i 960, Motorola 88110, IBM Power 6000 以及 Super SPARC。只有 MIPS 公司的 R 4000 属于超流水线结构。

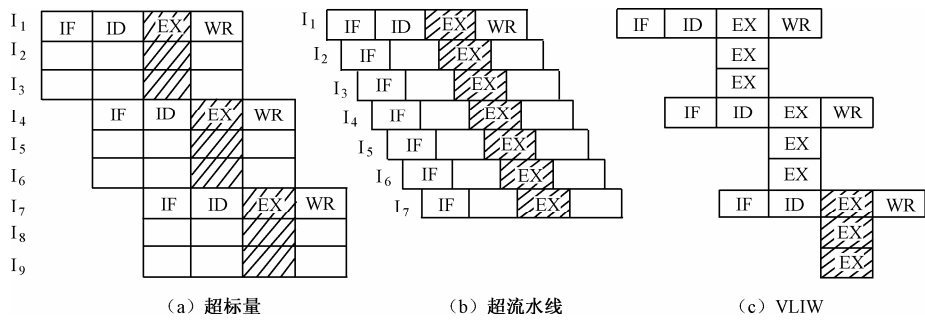


图 7-2 超标量、超流水线和 VLIW 流水线结构

VLIW 提出了很有启发性的新设计思想。在 IF 流水级，它取出的是一个长指令字（本例中是包含三条指令的长字），这个长指令字一起在 ID 流水级中译码，但在执行 EX 级时，分别由三个独立的执行部件执行。VLIW 最主要的问题是，它的指令格式特殊，不能做到与现有的处理器指令格式兼容。

在图 7-2 中，每个周期中发出的指令数为 3。理论上最佳情况为 $IPC = 3$ 。实际的新一代 RISC 产品，每个周期发出的指令数为 2~4，迄今尚无超过 4 者。以图中的多发射结构的例子来说，实际上每一个 IF 流水级（或周期）内取出的三条指令中都可以有数据相关与转移相关问题。而且本周期的三条指令与下一个周期内的三条指令之间也可能发生相关性问题。此外，由于“多发射结构”内含有多个执行部件，因此资源冲突的机会比单发射结构的 RISC 更为复杂。所以多发射结构中的指令调度任务更加重要，而且效果也可能更大。

7.3 指令调度

指令调度的目的是通过指令重组来提高指令级的并行性（ILP），使指令尽可能地并发执行。ILP 公式（对于单/多发射结构都适用）为

$$ILP = \frac{\text{程序执行总指令数}}{\text{程序执行总周期数}}$$

对于多发射结构处理器，ILP 可能大于 1。而加速比

$$SUF = \frac{ILP(\text{优化以后})}{ILP(\text{优化以前})}$$

现举一例说明指令调度后的并行执行。源程序经编译优化后可在 5 个步骤内完成。

源码	串行执行	调度后并行执行
1 C=A+B	OP1: LOAD A	1 OP1: OP2
2 K=I*J	OP2: LOAD B	2 OP5: OP6: OP3
3 L=M-K	OP3: C=A+B	3 OP9: OP7: OP4

- 4

Q=C/K
- OP4:

STORE C
- OP5:

LOAD I
- OP6:

LOAD J
- OP7:

K=I*J
- OP8:

STORE K
- OP9:

LOAD M
- OP10:

L=M-K
- OP11:

Q=C/K
- OP12:

STORE Q
- 4

OP8: OP10: OP11
- 5

OP12

指令调度是在程序经过初步编译，生成以助记符为基础的中间代码后进行的。

现代 RISC 系统的高级语言编译和优化全过程可用图 7-3 表示。其中，全局优化和中间编译等过程为过去传统的编译优化技术常用。而底层的局部优化和流水线窥孔优化是和 RISC 体系结构密切相关的。

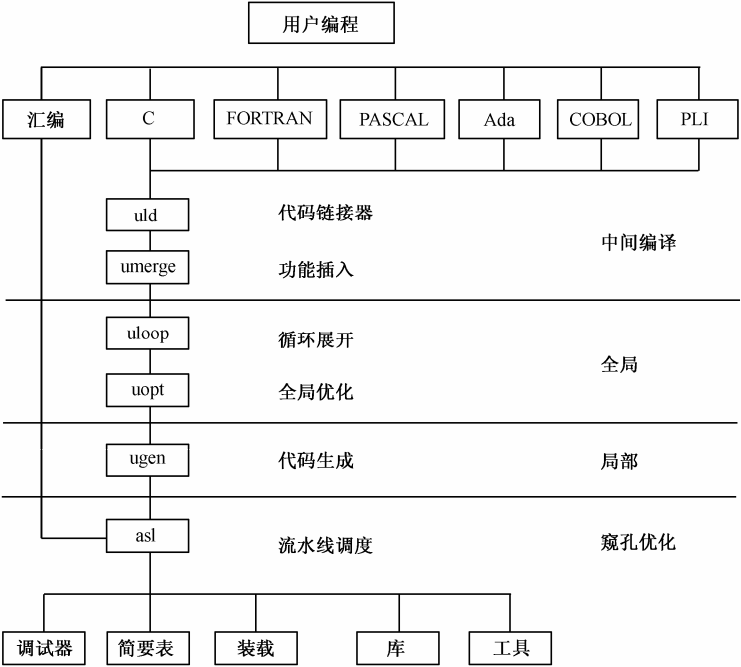


图 7-3 优化和编译全过程

用户程序在代码生成，产生以助记符表示的 asl 代码以后，就要根据具体 RISC 处理器的流水线结构以及资源冲突情况，进行指令调度。但是，整个程序的 asl 代码太长，必须在一个确定的范围内，才便于应用各种算法进行指令调度。一般采用基本块为单位来限定这个范围。

基本块的定义是只有一个入口点和一个出口点的代码段。该段中没有转移等分支，其流程图呈直线状。段头为入口点，段尾为出口点。

在一个基本块内进行指令调度，可以使问题简化，避免了转移相关的问题，只需解决数

据相关和资源冲突。

例如，数据相关可以较方便地用硬件或软件检测出来，然后在两条发生数据相关的指令中间，靠指令调度插入一条与上述两条指令无顺序关系但却有用的指令，这样就可以避免流水线的阻塞“气泡”，使流水线执行效率提高，而这种指令顺序的变换又不会影响程序执行的结果。又例如，资源冲突可以通过填写一个资源利用的表格来避免。记下每条指令是否使用该资源，以及使用资源的初始时间和结束时间，从而把可能在一段时间内同时使用某一资源的两条指令调度错开。

经分析统计，一般程序中的基本块所包含的平均指令数不多，通常只有 6~8 条指令。有时基本块内指令数峰值很大，达十几条或二十几条。但有时转移频频发生，一个基本块指令数就很小了。在这种情况下，指令调度所提高的 ILP 并不大。在目前的超标量 RISC 结构中，每个周期可发出的并行指令数不超过 4。所以，还必须考虑跨基本块之间的指令调度，或者把基本块设法“拉长”，才可以使 ILP 值提高较多。很明显，如果在调度范围内指令数越多，复杂的转移相关性处理需求越低，则指令调度的潜在效果越好。

另外，因循环在程序中出现的频度相当可观，如果把循环加以优化，也可使指令级并行性 ILP 更高。而循环优化也可在流水线调度级或全局优化级进行。经过这些综合方法的“指令级并行处理”，可以使多发射结构处理器中，经过指令并行处理的程序目标代码的执行速度比不经过处理的程序快 5~8 倍，有的甚至快十余倍。

指令级并行处理 ILPP (Instruction Level Parallel Processing)，近几年来已发展为一个专门的研究领域。它与传统的并行处理技术相辅相成。可以认为，传统的并行处理是解决系统中处理器之间的执行并行性问题，而指令级并行处理 ILPP 是解决处理器内部（尤其是多发射结构处理器）中功能部件之间的执行并行性问题。对于前者，已经存在的应用程序必须经专业人员修改而不能直接在传统并行处理系统中运行。尤其是并行编译技术尚未成熟时，它必然影响并行处理系统应用的局限性。指令级并行处理 ILPP 具有吸引力的地方是：已经存在的应用程序可以直接运行，在指令级上进行指令调度来提高指令执行的并行性，对于用户是透明的。目前，通过 ILPP 较好情况下达到的加速比为 3~8。当然，传统的并行处理系统可扩展性很高，处理器数目可达几千以至上万个，因而加速比可以非常高。但是，随着多发射结构功能部件的扩展以及指令调度（尤其是动态窗口的扩大）和编译优化技术的愈加成熟，ILPP 的加速比也可以有数量级的提高。总之，处理器之间的并行处理和处理器之内的并行处理，两者是应该互相结合的。

7.4 Cache结构

Cache 是一种小容量、高速度的存储器，用在计算机系统的处理器和主存储器之间，存放当前使用的主存的部分内容（副本），以减少访问主存的等待时间。

在页式虚拟存储系统中，Cache 可以有实地址访问或虚地址访问。页式虚拟存储器为程序提供较大的虚地址空间。程序使用虚地址，在运行时可以由系统负责把虚地址转换为实地址。这种虚存机制允许程序的容量不受实际内存容量的限制，而且也不要求实存空间必须有连续的地址。

系统使用页表把虚地址转换为实地址。虚地址用来访问页表中的某一页表项，从页表中可得到实地址。使用一个称为转换检测缓存（以下简称 TLB）的硬件，可以加速这种转换过程。一个 TLB 实际上也是一个小容量的 Cache，一般可以容纳 64~512 个当前使用的页表项。同所有的 Cache 一样，TLB 缩短了页表项的访问时间，从而减少了地址转换的开销。

7.4.1 实地址Cache

1. 实地址Cache的结构

图 7-4 所示是大多数计算机系统所采用的实地址 Cache 和 TLB 的原理图。处理器发出虚地址送到 TLB 及 Cache，并行地查找实地址及数据。TLB 中包含虚、实地址对。如果经虚地址比较相等，则从 TLB 中得到实地址。Cache 中包含实地址标志及数据，从 TLB 中得到的实地址与访问到的实地址标志比较，如果相等，则从 Cache 得到数据，这是 TLB 及 Cache 均命中的情况；如果实地址比较不相等，则发生 Cache 不命中，这时要到主存去查找页表，将取来的页表项送入 TLB。

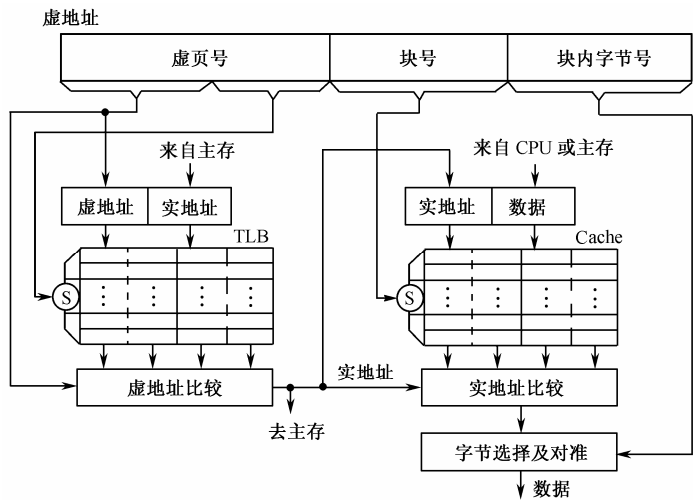


图 7-4 典型的实地址 Cache 及 TLB 原理图

虚地址空间较大的计算机系统中，页式虚存管理大多采用两级页表完成地址转换，如图 7-5 所示。例如，把 32 位虚地址（4GB）分成 1M 个 4KB 页，共需 1M 个页表项，每个页表项占 4B，共 4MB。对这样多的页表项也需进行页式管理。管理页表项的信息称为页目录或二级页表。如果把页表分成与数据大小相等的页，则可有 1K 个页表，所以需 1K 个页目录项，每项 4B，页目录共占 4KB。页目录基地址指出页目录在内存中的起始地址，它应该是事先定义好的。

2. 放置算法

放置算法为主存地址与 Cache 单元之间提供一种映像机制。常用的放置算法有全相联映像、直接映像以及组相联映像三种。一个 Cache 项至少要包括地址标志、一位有效位和一个数据块。地址标志实际上是若干位高地址，使用它足以判断相应数据块的所属。有效位指明该块是否为有效信息。

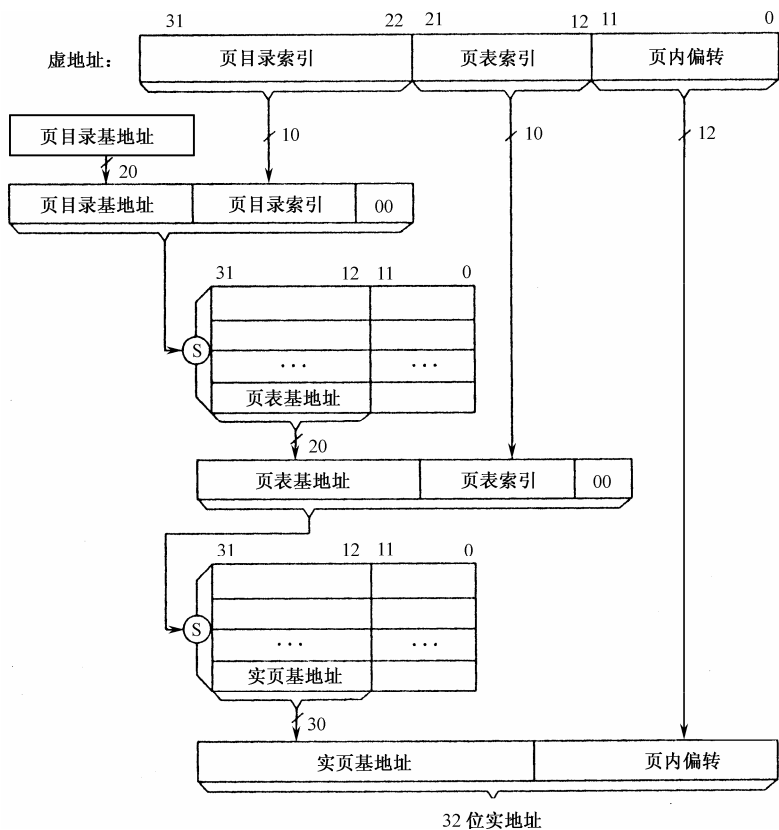


图 7-5 两级页表虚存管理

在直接映像方式下，给定的内存仅能放在一个特定的 Cache 块中。它使用若干低位地址直接访问 Cache 项，并把剩余的高位地址与 Cache 项中的地址标志进行比较。如果相等且有效位为 1，则一次命中发生。这是最简单的放置算法，它的优点是访问 Cache 的速度快，实现简单；缺点是当处理器轮流使用两个或多个块，而这些块又都映像到同一 Cache 项时，命中率会急速下降。

组相联映像也使用低位地址，但它选中的是一个组，组内包含两个或多个块。给定的内存块可以放在选中的组中任意一块内。一组内的块数，一般称为相联度或相联路数。选中一组后，组内所有的项标志同时进行比较，如果有一个匹配，则发生一次命中。组相联映像实际上是靠增加比较器的个数及增宽 Cache 块的位数来降低 Cache 块的冲突的。

在全相联映像方式下，给定的内存块可以放在 Cache 的任意一块内。它一般使用内容定址的存储器（简称 CAM）来实现。送来的地址与 Cache 所有项中的地址标志进行比较，以判断是否命中。尽管这种方法有较高的命中率，但 CAM 的访问速度较慢，而基于 CAM 的 Cache 还会占用较大的芯片面积。这是因为：① 每个 CAM 单元要比 RAM 单元大；② 基于 RAM 的 Cache 要求较少的标志位，其原因是每个 RAM 单元的地址并不需要存储。

3. 替换算法

当发生 Cache 不命中时，必须选择要被替换的项（当然，在直接映像方式下只有一种选择）。常用的方法有随机法和当前最少使用法 LRU。随机法容易实现，可以选择一个随机事

件作为替换的依据，比如，可以用系统实时时钟的低几位。LRU 法要维护一个各项使用的次序表（用链表或关系矩阵表示），次序可以是当前最多使用到当前最少使用的排列。当一项被访问时，则把该项移到当前最多使用的位置。这样，当需要替换时，总是选择当前最少使用的项。对于组相联映像的 Cache，LRU 方法只是对组内各项起作用。如果相联度超过 2，则实现起来要复杂一些。研究表明，替换算法对 Cache 命中率的影响不十分明显，因此多数系统使用最简单的替换算法。当 Cache 容量较大时，干脆采用直接映像方式，以消除实现替换算法的开销。

7.4.2 虚地址Cache

在实地址 Cache 中，每次对 Cache 的访问都必须完成地址的转换。在大多数系统中，要求地址转换与 Cache 访问并行进行，即要有足够的不需转换的位来访问 Cache。对于有固定大小的页的系统来说，增加 Cache 容量将会增加 Cache 的复杂性（即相联度）。例如，16KB 的 Cache 在 4KB 的场合下，必须实现 4 路组相联结构，即要满足“Cache 组数×块容量≤页容量”的要求。然而，对于较大容量的 Cache，增大相联度往往是不现实的，地址转换也只好与 Cache 的访问串行进行。

虚地址 Cache 可以避免转换延迟问题。它由虚地址低位作为索引访问 Cache，并把虚地址高位与 Cache 中的虚地址标志进行比较。其优点是，仅当 Cache 不命中时才进行地址转换。在页式虚拟存储器中，地址转换要用页表项。如果仍用 TLB 来加速地址转换，除了 TLB 利用率不高外，在多处理器系统中还存在着与 Cache 相类似的 TLB 一致性问题。在有些虚地址 Cache 的实现方案中，比如 Xerox 的 Dradon，整个系统中有一个 TLB，由所有处理器共享。这种方法解决了 TLB 的一致性问题的，但可能成为系统性能的瓶颈。

在虚地址 Cache 中不使用 TLB，有没有办法加快 Cache 不命中时的地址转换速度呢？下面介绍的 Cache 内地址转换便试图解决这个问题。仍以两级页表虚存管理为背景加以说明。这种方式的实质是把页表项同数据等对待，放在同一 Cache 中，并且也用虚地址访问它们。地址转换是从 Cache 中找页表项，而不是到 TLB 中去找。在 Cache 中查找页表项是在访问 Cache 不命中时才开始的，因此与 TLB 方法相比，这种机制引入了附加的延迟。好在这种转换仅在 Cache 不命中时发生，所以在性能方面造成的损失较小。

为了转换虚地址，需要取到它的页表项。由于页表项也要到 Cache 中去查找，因此必须计算出页表项的虚地址。这种计算必须能够快速完成，否则便失去了把页表项放在 Cache 中的意义了。为此规定所有页表项在虚地址空间是连续的，使用数据的虚页表号作为访问页表的索引地址。再进一步规定所有页表在虚地址空间是地址对准的，用简单的移位和拼接方法便可以得到页表项的虚地址，如图 7-6 所示。

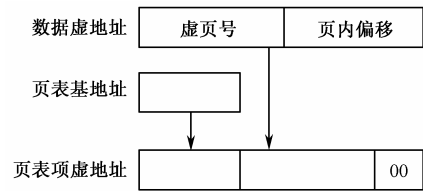


图 7-6 页表项虚地址的形成

图 7-7 给出了访问存储器可能出现的 4 种情况。A 是 Cache 命中的情况，它是最经常出现的。在访问 Cache 的同时，Cache 控制器中的移位拼接电路要形成页表的虚地址以备访问 Cache 不命中时使用。如果需要做地址转换，Cache 控制器使用这个地址试图到 Cache 中读页表项。B 是访问数据不命中而访问页表项命中的情况。与情况 A 相比，情况 B 除了多一次

Cache 访问外，还多一次取数据块的存储器访问。如果页表项也不在 Cache 中（情况 C），要求第 3 次 Cache 访问以取得页目录项，由此得到页表项的实地址。从存储器取来的页表项放入 Cache 中以备将来使用。在最坏的情况 D 下，所有三次 Cache 访问均失败，页目录项也需到存储器去取，取来的页目录项放入 Cache 中。页目录的基地址是事先规定好的可以保存在 Cache 控制器中。

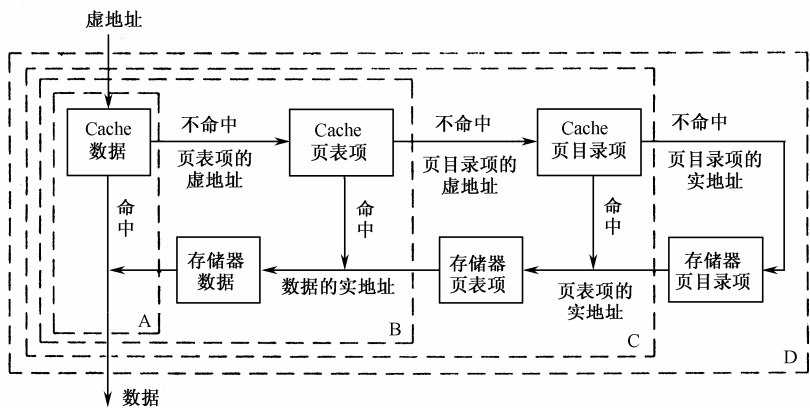


图 7-7 Cache 内地址转换机制

7.4.3 多处理器的Cache一致性问题

与别名问题不同的是，在多处理器共享存储器的环境下，如图 7-8 所示，各处理器使用相同的虚地址访问各自的 Cache 可能得到不同的数据。写直达策略会加重总线的负担，尤其是在虚地址 Cache 中，访问存储器需要增加一次对 Cache 的访问以得到页表项中的实地址，造成延迟增大。因此，在大多数虚地址 Cache 的多处理器系统中都采用写回策略。使用写回策略时，由于存储器的内容有可能不反映 Cache 中的最新内容，则一致性处理要比使用写直达策略时更为复杂。

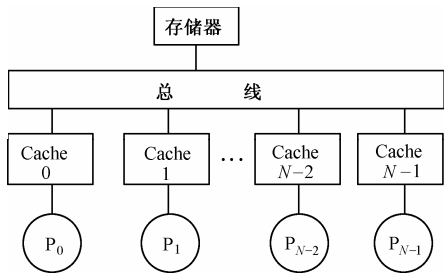


图 7-8 共享总线的多机系统

如果所有处理器共享一个 Cache，则一致性问题迎刃而解，但会造成系统性能瓶颈。在每个处理器都带有各自 Cache 的情况下，不使共享的且可写的数据进入 Cache，即只允许它们存在于主存中，也是解决一致性问题的办法。

以下是一种改进的办法。在每个 Cache 中另附两位状态位，主存不需要任何状态位。第 1 位指明该块是共享的还是独用的，第 2 位指明该块是否被本地处理器修改过。这种方法不允许 Cache 中出现已被修改过的共享块状态，因此用状态表示该块信息是无效的。可能的 4 种状态是：

- (1) 无效。该块不含有效信息。
- (2) 独用未改。其他 Cache 无该块，且数据块与主存一致。
- (3) 共享未改。其他 Cache 也可能有该块，数据块与主存一致。

(4) 独用已改。其他 Cache 无该块，数据块已被本地处理器修改过，因此与主存是不一致的。

当一个 Cache 块被替换时，仅状态独用已改的块才写回主存。如果使用写直达策略，则没有必要区分独用已改和独用未改两种状态。在写回策略下，对独用块的写操作要完成的仅是修改块内容并置独用已改状态。共享未改意味着其他 Cache 也可能有该块数据。开始置成该状态时至少必须有两个 Cache 包含该块。当然，除了后一个以外，其他 Cache 中的该块均被替换时，它便不是真正的共享块了，但为实现方便，该块状态便保留不改了。

如图 7-9 和图 7-10 所示为 Cache 读、写操作的大致过程。当 Cache 不命中时，一个读请求广播到所有的 Cache 及主存。如果是由写操作引起的不命中，伴随请求地址的还有一个废除命令。如果一个 Cache 与请求地址匹配，则它禁止主存送出数据。假设 Cache 操作是异步的，用一个简单的优先级电路便可解决多个 Cache 的响应问题，其中，较高优先级的 Cache 把数据放到总线上。如果所有 Cache 中均无请求块，则由主存提供。所有与请求地址匹配的 Cache 块把所有的状态置成共享未改。如果该块原来处在独用已改状态，则还要把它写回主存。对于写操作，匹配的 Cache 把匹配块置成无效状态，如果请求的数据由其他 Cache 提供，则请求者把该块置成共享未改状态。如果由存储器提供，则置成独用未改状态。

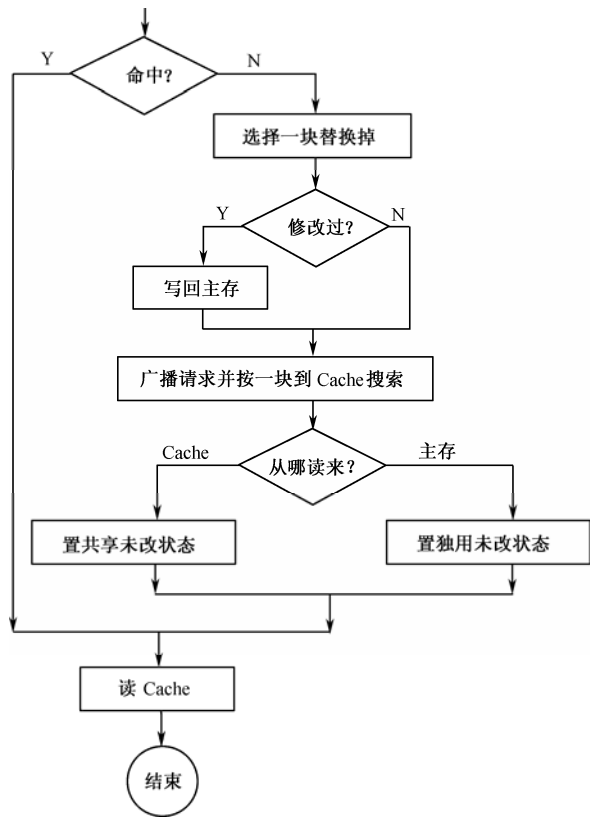


图 7-9 Cache 读操作

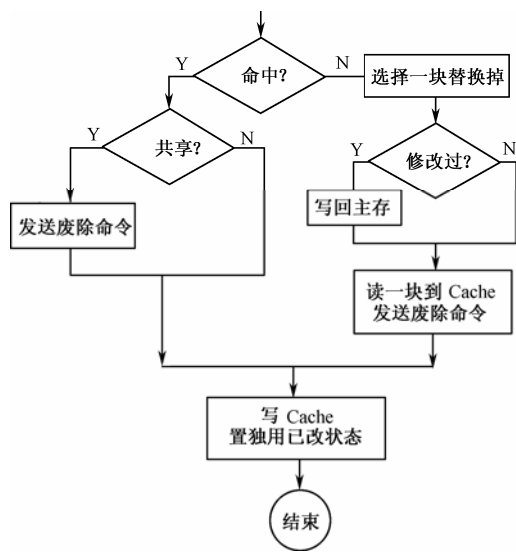


图 7-10 Cache 写操作

第 8 章 分布计算环境结构

分布计算环境（Distributed Computing Environment）是为网络计算平台提供开发工具和服务，以资源共享和协同工作为主要工作方式，以单一映像为用户实现分布式应用的系统结构。

8.1 分布计算环境的发展

从 20 世纪 80 年代中期开始至今，分布计算环境经历了两个阶段，当前正进入第三阶段。

第一阶段最典型的是以解决信息共享为目的的 Client/Server 结构。它向人们提出了在异构计算环境下的互操作、分布式操作系统等一系列在集中计算环境下不曾出现或并不突出的技术难题。在这一阶段，主要还是由用户运行传统的计算概念和设施，如过程调用、文件调用。

第二阶段是以面向对象技术为特征的多层 Client/Server 结构。将系统中的所有资源按对象来组织，创建和维护分布对象实体的应用称为 Server，按照接口访问该对象的应用称为 Client。Server 中的分布对象不仅能被访问，而且也可作为其他对象的 Client。在这一阶段，对象请求代理 ORB(Object Request Broker)技术是实现 Client 访问异地分布对象的主要手段。

第三阶段可以说分布计算环境正在与网格计算（Grid Computing）融合。这部分内容请参阅第 11 章的有关章节。

8.2 客户-服务器结构

客户-服务器结构（Client/Server）的概念最早用于描述软件的系统结构，表示一个应用程序与一个服务程序之间的关系。随着局域网和分布式系统结构的迅速发展，对客户-服务器的理解越来越复杂和深刻。目前，客户-服务器结构是分布式计算环境中采用最多的计算模型。它由 Client 提出处理请求，而 Server 为 Client 提供系统定义的各种服务，如各种基于文件的服务、数据库服务、名字目录服务、事物处理服务等。客户-服务器结构可以降低软件的开发和维护成本，增强应用的可移植性，改善网络 and 系统的性能，提高用户的工作效率，保护用户的投资，减少对小型计算机和大型计算机的需求。

8.2.1 客户-服务器结构的特点

客户-服务器是分离的逻辑实体，它们通过网络协同来完成一项工作，该结构具有下列特点：

（1）Server 进程是服务的提供者，Client 进程是服务的消费者，它们在功能上是分离的，是可以在不同的机器上运行的进程间的一种关系。

（2）Client 只是向 Server 提出要求处理的请求和相关内容，Server 将最终处理结果回送给 Client。在整个服务过程中，它们之间并没有传输整个处理程序和数据文件，提高了网络的利

用率，减小了网络流量。

(3) 一个 Server 可以同时为多个 Client 提供服务；同样多个 Server 可以为一个或多个 Client 提供服务。而且 Server 的位置对 Client 而言是透明的，充分实现了资源的共享。

(4) 通过把应用程序同它们处理的数据隔离，可以使数据具有独立性。未通过鉴别和授权的用户将无法对数据进行非法访问，保证了系统数据的完整性和服务器对数据存取的有效控制。

(5) 支持系统的扩展性。可以在 Client 端增加或更换客户工作站或 PC，在 Server 端增加新服务器或转移到别的服务器上。

8.2.2 中间件的概念和特点

客户-服务器结构中采用中间件（Middleware）来提供平台与应用之间的通用服务。它是一个独立的软件层，具有标准的程序接口和协议，从而避免了应用系统与具体平台之间的紧耦合。针对不同的操作系统和硬件平台可以有符合接口和协议规范的多种实现方式。在客户-服务器结构里，一般将中间件放在 Client 和 Server 之间的中间层，负责应用逻辑的处理，从而为 Client 端“减肥”。这种采用中间件的三层客户-服务器结构，如图 8-1 所示。

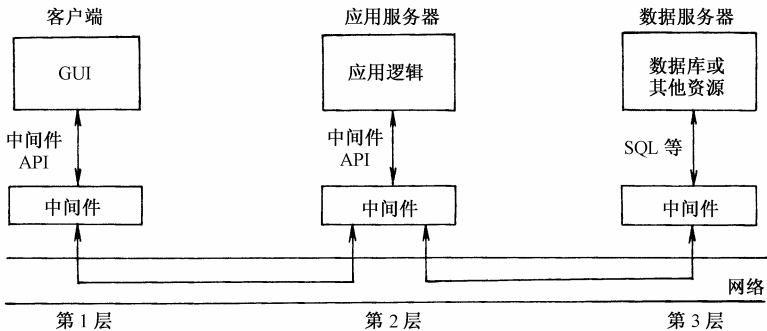


图 8-1 中间件在三层客户-服务器结构中的应用

中间件的特点如下：

- ① 中间件在工业界是通用的，能满足各行业的广泛应用。
- ② 中间件有多种版本，能在不同的平台上运行。
- ③ 中间件服务是分布的，它能支持分布对象计算、屏蔽网络和系统分布的复杂性。
- ④ 中间件服务支持某种标准协议。
- ⑤ 中间件服务支持标准化应用编程接口 API（Application Program Interface）。

8.3 开放式分布处理

开放式分布处理（Open Distributed Processing, ODP）是通过建立大家都能遵循的开放式标准，将物理位置不同的异构的信息系统组织起来，实现信息交互的新的系统结构。因此，建立开放式标准尤为重要。开始大家对建立标准的兴趣在于，建立一个使异构的信息系统能进行跨平台信息共享的公共系统结构，于是制定了标准化应用编程接口（API），以提高软件

的可移植性。同时，希望建立支持互操作的标准，以实现系统间对数据和程序的相互存、取。各个组织或厂商出于自身的考虑，出现了如 DEC，CORBA，NAS 等许多标准。20 世纪 90 年代初，出现了开放式分布处理框架。它着重研究构件的概念及其标准，系统的构成成分及其接口，接口的标准化，基于接口规范的“交易”和“联编”服务，任意配置的应用构件之间的逻辑关系（互操作），对分布的构件进行集成，对分布式平台实现可移植性，对应用程序屏蔽分布环境的细节，等等。ODP 推出的第 1 个标准化文本是开放式分布处理参考模型 RM-ODP（Reference Model-Open Distributed Processing）。它在统一开放式分布处理领域的各种标准方面起到了重要作用，有人称它为“标准的标准”。RM-ODP 是为分布式处理提供的一个通用的系统结构框架，将开放式思想引入分布式处理，以解决异构系统之间各种相关机制的一致性问题。

- ODP 的重点研究内容包括：
- ① 构件是用于复用的软件实体，根据复用阶段、复用方式的不同，与之相对应的构件表现形式也不同。
 - ② 系统的构成成分及其接口。
 - ③ 接口的标准化。
 - ④ 基于接口规范的“交易”和“联编”服务。
 - ⑤ 任意配置的应用构件之间的逻辑关系，实现互操作。
 - ⑥ 对分布的构件进行耦合，使之能提供某种服务，实现集成。
 - ⑦ 对分布平台标准化，实现可移植性。
 - ⑧ 对应用程序屏蔽分布环境的细节，实现透明性。
 - ⑨ 针对不同媒体的应用采用一致的建模框架。

8.4 公共对象请求代理体系结构

公共对象请求代理体系结构（Common Object Request Broker Architecture，CORBA）是由对象管理组织（Object Management Group，OMG）制定的一个工业标准。

OMG 先制定了对象管理体系结构（Object Management Architecture，OMA），如图 8-2 所示。该结构的核心是对象请求代理（Object Request Broker，ORB），它定义为在异构环境下对象透明地发送请求和接收响应的基本机制，负责实现客户机与对象之间的通信。CORBA 标准就是为 ORB 制定的规范。

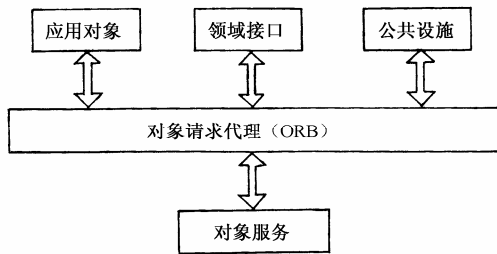


图 8-2 OMA 参考模型

CORBA 的核心思想是采用标准的接口定义语言，将软件接口与软件实现相分离。为了以统一的方式来描述对象接口，引进了接口定义语言 IDL（Interface Definition Language）及其映射。使用 IDL 语言的用户可以根据 IDL 接口中的信息来决定如何发出请求和接收响应，使用户对象完全独立于具体对象实现所在位置、使用的编程语言。

CORBA 通过对象系统为客户提供服务，

对象间的交互通过 ORB 传递。在 CORBA 中，对于一个交互来说，有客户方和对象实现方。客户通过发出一个请求来申请得到服务，与请求相关的信息包括操作、目标对象、几个参数以及请求上下文等。对象的实现使用接口来描述对它执行的操作。ORB 将请求交付给目标对象并返回响应给发出请求的客户。其结构如图 8-3 所示。

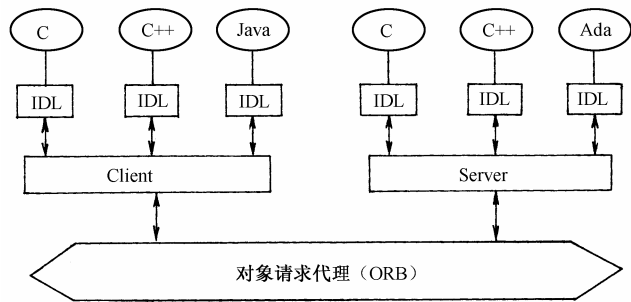


图 8-3 采用 CORBA IDL 提供客户-服务器互操作性

基于 ORB 的基本开发过程如下：

- ① 定义 IDL。对于系统中需要利用 ORB 进行交互的对象，首先应对其对外呈现的接口进行定义。
- ② 将 IDL 映射为具体语言的 Stub/Skeleton。IDL 是独立于具体编程语言的，而应用程序则一定要用具体语言完成。因此必须将 IDL 进行映射，产生由具体语言表示的接口，以供调用。
- ③ 编写实现具体服务功能的代码。ORB 提供的仅是对象间互操作的支持，至于对象的功能，则必须由编程者实现。
- ④ 编译、连接，产生服务器程序。
- ⑤ 编写调用具体服务功能的代码。
- ⑥ 编译、连接，产生客户程序。

8.5 基于Web的分布计算

在基于 Web 的分布计算模式下，用户通过浏览器（Browser）把网络中心计算模式带到了任意一台 PC 上，Web 页面及图像事实上都来自 Web 服务器。作为新一代的计算技术，Web 能在不同的网络及操作系统中运行，并能方便地扩充到外部的相关用户，它是一种完全通用的以服务为中心的系统结构。

8.5.1 Browser/Server结构

基于 Web 的分布计算模式是建立在客户-服务器结构上的。Web 服务器用来储存供 Browser（浏览器）浏览的页面，其内容除了标题和正文外，还包括同其他页面链接的信息等；用户通过浏览器向分布在网络上的许多服务器提出服务请求，Web 服务器向浏览器发送 HTML（超文本标记语言）文件，浏览器阅读、识别该内容，然后显示出页面；在 TCP/IP 上使用 HTTP（超文本传输协议）进行文件传输。

Browser/Server 结构简化了客户端的管理工作，只需安装和配置少量的客户端软件，对数据库的访问和应用系统的执行将在服务器上完成。服务器分布在网络上，用户不必关心提供的服务来自哪个服务器，而把这些服务器当作一台“虚拟主机”。因此，Browser/Server 结构是在客户-服务器结构的基础上扩充而成的，也称“瘦”客户机“肥”服务器模式，常称为“三层 Browser/Server 结构”，如图 8-4 所示。Browser 层通过 Java Applet, JavaScript, ActiveX 等技术来处理客户端事务；中间层通过 CGI（Common Gateway Interface）等中间件负责接收本地或远程的处理请求，然后运行服务器脚本，把处理请求通过 ODBC 发送到数据服务器以获取相关数据，再把结果数据转化成 HTML 及各种脚本回送给客户端的 Browser。这里的数据库服务器负责数据库的管理、数据的处理和更新、数据的查询和传输等工作。

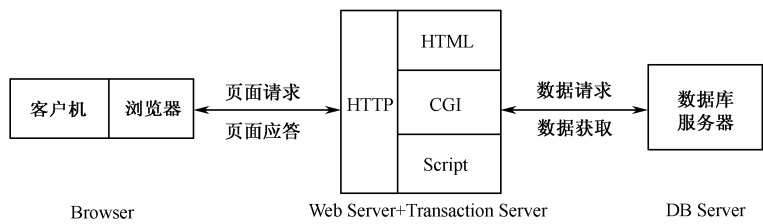


图 8-4 三层 Browser/Server 结构

Browser/Server 结构的特点如下：

- ① 它是由国际标准化组织制定的开放的标准。
- ② 由于在客户端工作比较少，投资低，因此安装、配置、升级等工作大多在服务器端，相比客户-服务器结构而言，开发和管理成本都很低廉。
- ③ 各种用户都可以用浏览器通过网络自由地访问系统。
- ④ 用户容易掌握浏览器技术，可以方便地使用系统。

8.5.2 基于Web的页面描述语言标准XML

基于 Web 的技术发展分为静态页面浏览和动态交互操作两大阶段。早期都采用静态页面浏览，它是由 Web 服务器使用 HTTP 协议将 HTML 文档从 Web 服务器传送到用户的 Web 浏览器上。HTTP 是一种以页面为传输单位、适合组织各种静态文档元素，如图片、文字等的基础协议。

到 20 世纪 90 年代中期，随着三层 Browser/Server 结构、CGI 标准、动态 HTML、Java/JDBC 等技术的涌现，产生了可以与用户动态交互的 Web 技术，大大丰富了基于 Web 的分布计算模式的实现。这里仅简单介绍最新的页面描述语言标准——可扩展标记语言 XML (Extensible Markup Language)。

XML 语言是一种原语言，着重描述数据内容的组织结构。通过这些组织结构信息可以引导不同的数据使用者将其关心的数据内容提取出来并用于各自的目的。XML 语言包含一组规则，用户可以根据这组规则自己来创建标记语言。这些规则使用一个程序解释器就能处理所有这些语言。XML 改变了浏览器显示、组织、检索信息的方式，它只定义文档的结构而未定义浏览器应该怎样解释一个文档，给 Web 开发带来了灵活性。为了提高数据查找效率，XML 还有一个关于文档数据组织结构的文档类型说明 DTD (Document Type Definition) 模式。用

户通过建立各种不同的文档类型定义，就拥有了不同格式的数据文件。

XML 语言的特点如下：

- ① 提供了一种功能强大、灵活高效地表达数据内容的方法。
- ② 通过定义 DTD，具有很强的可扩展性和自解释性，可被不同的程序用于不同的目的。
- ③ 在 XML 中，数据内容与具体应用无关，使用它表示的数据具有很好的使用效率和可重用性。
- ④ XML 文档是纯文本，可以用文本编辑器或可视化开发环境的任何工具创建和编辑。
- ⑤ 它将数据与显示分离，同一数据可以指定不同的格式进行显示。
- ⑥ 它对格式定义很严格，并有层次结构，处理容易。
- ⑦ 它有很强的链接能力，可以有双向链接、多目标链接、扩展链接和两个文档间的链接。

第 9 章 数据流计算机结构

为了设计高性能计算机的系统结构，一个方法是突破冯·诺依曼型结构，采用数据流方式，从而形成了数据流计算机。冯·诺依曼型计算机的基本特点是在程序计数器的集中控制下顺序执行指令，因此它是以控制流（Control Flow）方式工作的。美国 MIT 实验室的 Jack Dennis 及其助手于 1972 年首先提出了数据流模型，并证明由此而设计的数据流计算机，其性能价格比高，较好地切合了工艺技术的进步，能较方便地在应用领域中实现可编程应用。

9.1 数据流计算机的基本原理

传统的冯·诺依曼型计算机与数据流计算机相比，其工作原理根本不同。前者是在中央控制器的控制下顺序执行指令，而后者是在数据的可用性控制下并行执行指令。数据流计算机里没有程序计数器，其指令执行靠数据记号（数据令牌，Data Token）的可用性来实现，也就是指令的执行由数据来驱动，把控制流变为数据流。数据流计算机里没有常规的变量概念，也不存在共享数据单元的问题，程序顺序性仅由指令内部数据相关性控制，也就是，只有操作所需要的数据可用时，才启动指令执行（异步性）。它的所有操作都具有函数性，即所有指令都可以按任何顺序并行执行。正是这些特性，使数据流计算机中许多指令可以同时异步执行，预计隐含的并行度是很高的。

总之，在数据流计算机中，当指令所需数据可用时，该指令即可执行。这说明，指令的操作不受其他控制的约束。任何一条指令只要它所需要的数据齐全，且可用时都可以执行。数据流计算中没有变量的概念，也不设置状态，在指令间直接传送数据，因此操作结果不产生副作用，不改变机器的状态，从而具有纯函数的特点。由此可见，数据流计算机有如下优点：第一，对指令来说，摆脱了外界强加于它的控制，多条指令在数据可用性驱动下可同时并行；第二，它可以直接支持函数语言，不仅有利于开发程序中各级的并发性，而且有利于改善软件环境，提高软件的生产力。

9.2 数据流计算机的指令

9.2.1 数据流计算机指令的组成

在数据流计算机中，一条指令包含操作包（Operation Packet）和数据令牌（Data Token）两部分，如图 9-1 所示。

操作包（或指令包，Instruction Cell）通常由操作码、源操作数、后继指令地址组成，又可以看作是由操作型和受处理单元影响的部分两者组成的。操作型包括操作码、有关数的指示及常数值、目标操作地址、本操作要求几个数据令牌的标志等信息，这些信息一旦设定后就不再改变。受处理单元影响的部分则包括已经收到的操作数值、数据令牌已到标志、正在等待的数据令牌等信息。操作包在存储器中将占据一定大小的空间。

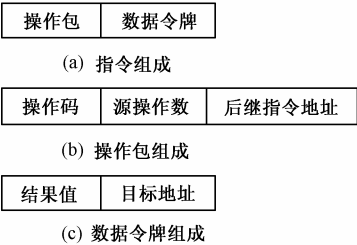


图 9-1 数据流计算机指令组成

数据令牌通常由结果值和目标地址组成。数据令牌的实质是一种表示某一操作数或参数已准备就绪的标志。一旦执行某一操作的所有操作数令牌到齐，则标识出这一操作是什么操作，以及操作结果所得出的数据令牌将发送到哪些等待此数据令牌的操作的第几个操作数部位等信息。所有这些信息都将作为一个消息包（Message Packet）传送到处理单元或操作部分予以执行。这样的消息包也称操作包。

9.2.2 数据流计算机指令的执行

在数据流计算机中，用数据令牌传送并激活指令，用一种有向图表示数据流程图。一条指令由一个操作符、一个或几个操作数及后继指令地址组成。后继指令地址也可以有几个，它的作用是把本指令的执行结果送往需要它的指令中。例如， $x = (a+b) \times (a-b)$ 这个函数中，其数据流程图如图 9-2 所示。为了表示数据在程序图中的流动状态，利用实心圆点代表令牌沿弧移动。假设 $a = 8, b = 12$ ，则图 9-3 所示通过令牌沿弧移动的先后过程反映出数据流程图图的执行过程。实际上，实心圆点代表该输入数据已准备就绪，旁边的数字代表此数据值。图 9-3（a）表示初始数据就绪，激发（驱动）复制结点以复制多个操作数；图 9-3（b）表示复制结点驱动结束，激发数据已准备就绪的+、-结点；图 9-3（c）表示+、-结点驱动结束，激发数据已准备就绪的*结点；图 9-3（d）表示*结点驱动结束，输出计算结果。

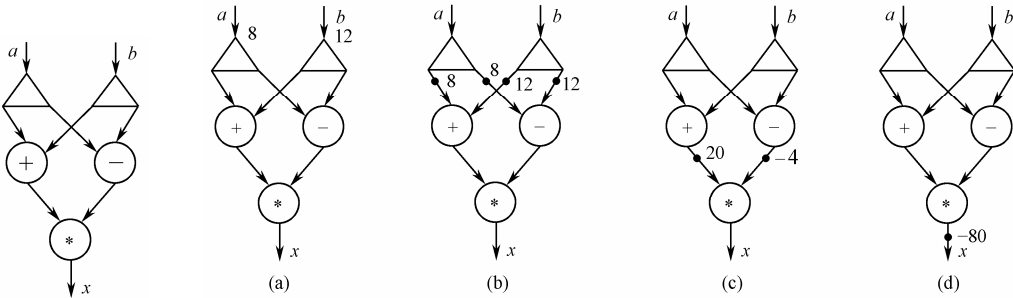


图 9-2 数据流程图

图 9-3 数据流程图图的执行过程示意图

总之，数据流驱动的结果，具有十分明显的 4 个特性。

- (1) 异步性（Asynchrony）。只要本条指令所需要的数据令牌都到达，指令即可独立执行，不必关心其他指令及数据的情况如何。
- (2) 并行性（Parallelism）。可同时并行执行多条指令，而且这种并行性通常是隐含的。
- (3) 函数性（Functionalism）。由于不使用共享的数据存储单元，所以数据程序不会产生

生诸如改变存储字这样的副作用（Side Effect），也可以说数据流运算是函数性的。

（4）局部性（Locality）。操作数不是作为地址变量，而是作为数据令牌直接传输，因此数据流运算没有产生长远影响的后果，运算效果具有局部性。

综上所述，数据流运算具有异步性、并行性、函数性、局部性，所以它很适合采用分布方式的计算实现，从而可以把数据流计算机看作是一种分布式或多处理机系统。

9.3 数据流计算机结构

根据数据令牌处理方式的不同，通常把数据流计算机的结构分为静态和动态两类。

9.3.1 静态数据流计算机模型及其结构

静态数据流计算机的主要特点是数据令牌没有标号。为了保证正确无误地工作，在任意给定时刻，当数据流程图中的结点操作时，其任何一条输入弧上只允许存在一个数据令牌。在静态数据流计算机中，数据令牌是沿数据流程序的弧流向操作结点。当所有操作数据都出现在输入弧上时，开始执行结点操作。任何时候只允许一个操作数呈现在一条弧上，否则后继指令不能得到区分。

操作过程是，数据令牌处于“更新部件”的输入池中，它将数据传递给“存储部件”的目标指令。凡已接收到全部所需数据令牌的指令都被“取指令部件”取出，加入到多条可执行指令的队列中，等待分派程序把它们按“先进先出”方式分配给处理部件中相应的空闲处理机并发执行。各处理器产生的新的数据令牌形成结果操作包（也称信息包），送到“更新部件”的输入池中，再由“更新部件”将这些数据令牌按它们的目的地址，送入“存储部件”指令池内相应指令的有关部位。同时，“更新部件”将已收到的全部必需的数据令牌的指令地址又“传输”给“取指令部件”，实际上它是以控制令牌的方式把这些已具备激活条件的指令地址送到“取指令部件”。典型的静态数据流计算机基本结构如图 9-4 所示。

MIT 的 J.B.Dennis 等人是数据流计算机研究的开拓者，他们所提出的数据流计算机系统的主体结构如图 9-5 所示。

（1）存储部件（Memory Section，MS）由若干指令单元组成。每个指令单元保存数据流

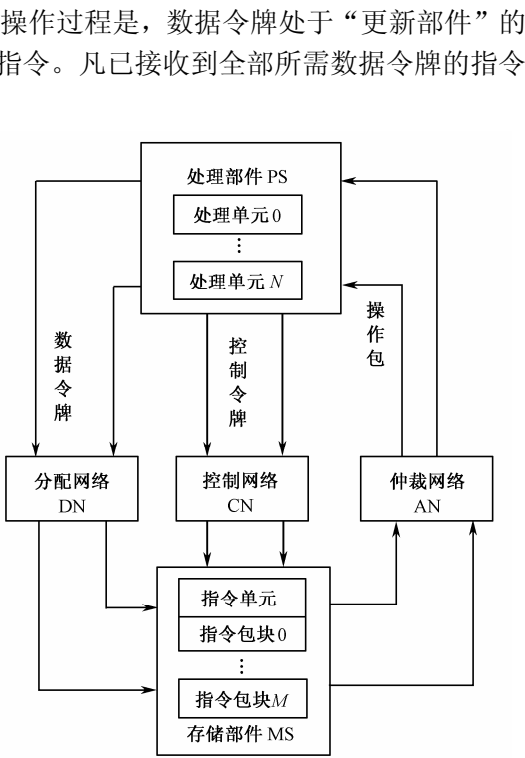


图 9-5 MIT 静态数据流计算机的基本结构

程序中的一条指令，它与数据流程图中的结点对应且由唯一的地址所指明。

(2) 处理部件 (Processing Section, PS) 由对数据值进行基本运算的多个处理单元组成，可以并发执行已被激活的指令所要求的操作。

(3) 分配网络 (Distribution Network, DN)。它将处理部件产生的多个结果数据令牌依据其各自的目的地址分别传输到存储部件相应的指令单元中。

(4) 控制网络 (Control Network, CN)。它将控制令牌由处理部件发送到存储部件相应的指令单元中。

(5) 仲裁网络 (Arbitration Network, AN)。它将可执行的操作包由存储部件发送到处理部件，同时允许有多个操作包在多个通路上传输。

9.3.2 动态数据流计算机模型及其结构

动态数据流计算机的主要特点是让令牌带上标记，它允许任意时刻在数据流程图任一条弧上出现多个带不同标记的令牌。因为令牌的标记是能识别该令牌时间先后关系的标号，所以不需要像静态数据流计算机那样用控制令牌来对指令间数据令牌的传输加以认可，这种方法能开拓程序的最大并行性。如果程序是循环的，则该标记方法允许动态无拘束地进行迭代计算。

动态数据流计算机的同步是由匹配机构实现的，数据令牌形成“匹配部件”的输入池，“匹配部件”完成数据令牌配对或成组，并将它们临时存储在某个存储空间，直到所有操作数得到比较。之后已匹配的指令组释放到“更新部件”，形成执行指令，放入可执行队列中，然后执行。

典型的动态数据流计算机的基本结构如图 9-6 所示。

动态数据流计算机系统的指令执行与“匹配部件”同步。“处理部件”中各处理单元输出的数据令牌送入“匹配部件”的输入池，“匹配部件”将这些数据令牌打上标记，并配成对或组，暂时保存起来，直到指令的一对或一组数据令牌全部配齐而被激活，送往“更新/取指令部件”。每组数据令牌是执行某条指令所需要的，一般二元操作需要两个数据令牌。“更新/取指令部件”取出这些激活的指令加入已配齐的操作数，然后再送往相应的可执行的指令队列。由“匹配部件”给每对或每组配齐的数据令牌加上不同的标记，以区别出迭代执行的不同层次，从而可以将迭代循环展开，实现对它的并行处理。

目前动态数据流计算机有网络状和环状两种。这里介绍 MIT 的动态数据流计算机的基本结构，如图 9-7 所示。它由 N 个处理元件 PE (Processing Element) 和一个实现 PE 间通信的 $N \times N$ 的包交换开关网络组成。每一个 PE 基本上是一台完整的计算机，主要包括程序/数据存储器、I 结构存储器、等待匹配部件、取指令部件、执行部件及其他硬件，如图 9-8 所示。

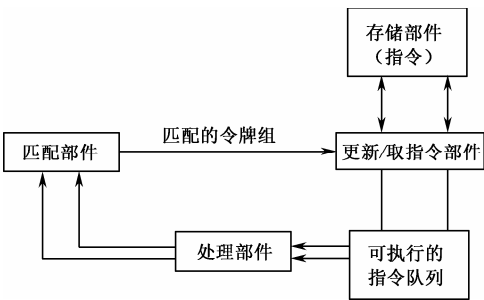


图 9-6 动态数据流计算机的基本结构

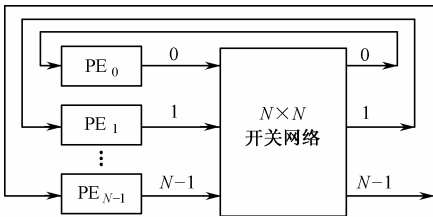


图 9-7 MIT 动态数据流计算机的基本结构

(1) 输入口有一个寄存器，此寄存器空闲时可以接收一个其他 PE 经开关网络送来的令牌，或接收一个来自本 PE 输出口的令牌。

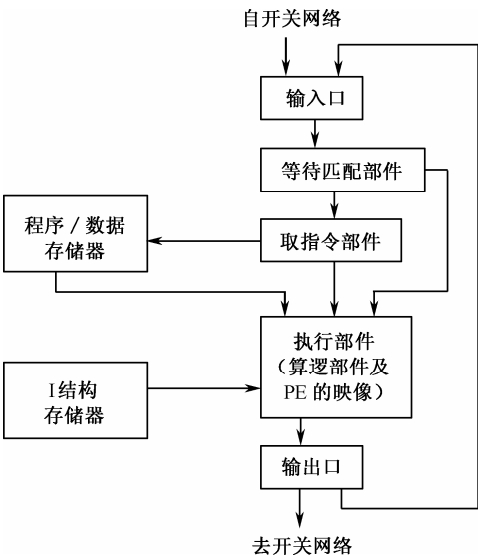


图 9-8 MIT 动态数据流计算机中 PE 的内部结构

复制操作，节省大量存储空间和辅助操作开销。执行部件除处理 I 结构的读操作外，也完成存储操作。

(4) 在 ALU 或存储器产生结果配上新的标记和目的地 PE 号后，送往输出口。通过输出口的开关网络送往目标 PE 或不经开关网络就直接送入本 PE 的输入口。

在 PE 各部件中，为了避免在多个通路传输数据时发生冲突、阻塞或死锁，各个部件都设有相应的缓冲器。PE 中没有程序计数器，但有一个存放已激活指令的列表，使已激活指令的执行是无序的。

9.3.3 静态与动态两种数据流计算机的比较

静态数据流计算机与动态数据流计算机的相同点是：有多个处理部件，可以独立、异步地执行多条指令，都依靠数据令牌来传输操作数和激活指令。两者的不同点在于，它们采用了不同的通信方式和同步方式。

在图 9-4 所示的静态数据流计算机中，数据令牌开始存放在“更新部件”的输入缓冲器内，通过该部件传送给指令“存储部件”。在指令“存储部件”中按照数据令牌本身携带的目的地址，把令牌中的操作数送到目标指令。当一条指令所需数据令牌全部到齐后，该指令开始执行。指令由“取指令部件”从指令“存储部件”中读出，并送到可执行的指令队列。一旦处理部件有空闲，该指令可即刻在“处理部件”中被执行。指令执行后把产生的结果与后继地址一起组成新的数据令牌，送入“更新部件”的输入缓冲池中，由此完成一条指令执行的全过程。

比较图 9-6 与图 9-4 不难发现，动态数据流计算机比静态数据流计算机多了一个匹配部件。在图 9-6 中，数据令牌首先存入“匹配部件”的输入缓冲器内，该输入缓冲器采用相联

(2) 等待匹配部件将已到的令牌暂时保存在缓冲器中，直到带匹配标记的另一个令牌到来。当所需要的两个数据令牌均已到达，且标记匹配时，就将它们转送到取指令部件的缓冲器中。根据标记中的指示说明，从程序/数据存储器中取出相应指令，形成包含操作码、操作数和目的地的操作包，送到执行部件予以执行。执行部件包含一个浮点算逻部件和一个用于确定一个新的操作名和目的地址该用哪个 PE 的硬件（即 PE 的映像）。

(3) I 结构存储器是一个带标记的专用存储器，用于存放类似数组的数据结构。如果操作数是结构数据，则该数据从 I 结构存储器中取得。I 结构每一个元素有一位标志，当读出时，若该元素的标志位为 0，表示无值，就自动将读操作推迟。使用 I 结构可以避免过多的数值

方式访问。“匹配部件”把缓冲器内的令牌配成对（或配成组）。例如，执行一个二元运算，通常需要两个数据令牌相符合，并把配成对的数据令牌集送“更新/取指令部件”。“更新/取指令部件”将送来的数据令牌对与使用该令牌对的指令相符合，并把执行指令所需要的操作数代入指令，形成可执行指令，送入可执行指令队列等待执行。只要某个处理部件有空闲，该指令即可执行。

由于动态数据流计算机的每个数据令牌上都有特殊标记，因此，可以通过“匹配部件”把一个循环程序的不同次循环同时展开进行迭代，从而大幅度提高程序的指令级并行度，缩短程序的执行时间。从原理上分析，动态数据流计算机能够更好地开发程序的并行成分，而且，由于中间运算结果直接代入下一条指令，不像静态数据流计算机那样，要写入内存或通用寄存器进行周转，从而减少了开销，提高了程序执行的速度。但是动态数据流计算机结构复杂，匹配部件中相联存储器设计比较困难，因此匹配部件通常成为影响并行度提高的一个瓶颈。

9.4 数据流程图和数据流语言

9.4.1 数据流程图

数据流计算机的程序是用数据流语言编写的，其机器语言也就是数据流程图。数据流语言的主要目标是开发程序内隐含的并行性，便于程序设计，自然表达程序中的并行性，以及运行的高效率。

数据流语言是函数语言，在执行前需要翻译成数据流程图（机器语言级程序），它执行的是所谓的点火原则，即一个操作可以点火的前提是：它的所有输入值全部到达，操作开始进行，将输入值吃掉，产生输出数据。数据流是有向图，结点对应操作符，弧是数据令牌迁移的指针。

J.B.Dennis 和 J.E.Rumbaugh 等提出了用于数据流程图的各种符号（即结点），并规定了相应的操作执行规则，现分别介绍如下。

1. 复制结点

图 9-9 示出了两种最常用的复制结点及其操作规则。其中，图 9-9（a）是数据复制结点（为实心结点），数据令牌 X 经过复制结点时执行复制操作，产生两个相同的数据令牌 X_1 和 X_2 。图 9-9（b）是逻辑复制结点（为空心圆点），控制令牌 X 经复制后产生两个相同的控制令牌 X_1 和 X_2 。

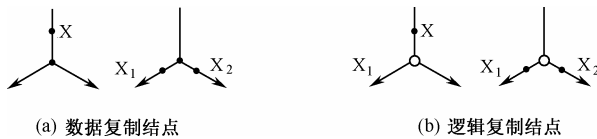


图 9-9 复制结点及其操作规则

2. 运算结点

根据操作功能，运算结点分为算术运算结点和逻辑运算结点。算术运算结点有 \oplus ， \ominus ， \odot ， \oslash ， \oplus ， \ominus 等。逻辑运算结点有 \wedge （与）、 \vee （或）、 \neg （非）、 \oplus （异或）等。按照输入端个数可把结点分为单输入结点和多输入结点两种。算术运算结点的箭头为实心，逻辑运算箭头为

空心。如图 9-10 所示是运算结点的几个例子。

3. 常数发生器结点

常数发生器结点无输入线，只有一条输出线，其功能是产生一个常数。图 9-11 (a) 是该类结点的示意图。图 9-11 (b) 是常数 2 的结点及其执行操作后产生数据令牌的情况，此令牌带有常数 2。

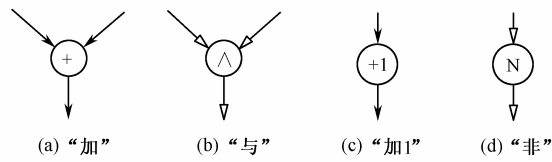


图 9-10 运算结点示例

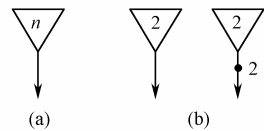


图 9-11 常数发生器及其操作

4. 条件分支结点

条件分支结点用于控制数据令牌传送时刻，通过空心箭头表示输入分支线，它输入的是控制令牌。常用分支结点有 4 种：

if true then $X \rightarrow X_T$
if false then $X \rightarrow X_F$
if true then $X \rightarrow X_T$ else $X \rightarrow X_F$
if false then $X \rightarrow X_F$ else $X \rightarrow X_T$

这 4 种条件分支结点如图 9-12~图 9-15 所示。

图 9-12 是 T (True) 门分支结点及其操作规则示意图。当输入分支线上的数据令牌 X 和带有真值的控制令牌 T 都到达该结点，且输出线上没有数据令牌时，该分支结点操作立即执行，将输入分支线上的数据令牌原样传送到输出线上。图 9-13 是 F (False) 门分支结点及其操作规则示意图。其工作过程与 T 的分支结点类似，所不同的只是输入分支线上的数据令牌 X 和带有假值 (False) 的控制令牌 F 都到达该结点，且输出线上无数据令牌时，该分支结点才执行操作。

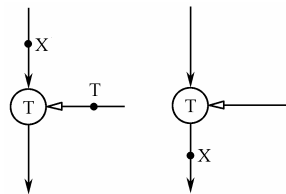


图 9-12 T 门分支结点及其操作规则

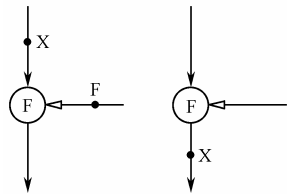


图 9-13 F 门分支结点及其操作规则

开关分支结点用椭圆表示，如图 9-14 所示。开关分支结点的操作执行规则是：当输入分支线上出现数据令牌 X 时，若控制输入分支线上的令牌是真值 (True)，且 T 输出线上无数据令牌时，该结点执行结果是把 X 送到 T 输出线上，如图 9-14 (a) 所示；相反，若控制输入分支线上令牌是假值 (False)，且 F 输出线上无数据令牌，该结点执行结果是把 X 送到 F 输出线上，如图 9-14 (b) 所示。

图 9-15 所示是合并分支结点及其操作执行规则示意图。它根据到达的控制令牌带的是真值 T 还是假值 F，决定是把 T 输入线上的 X 还是把 F 输入线上的 X 送到输出线上。因此，其作用正好和开关分支结点相反。

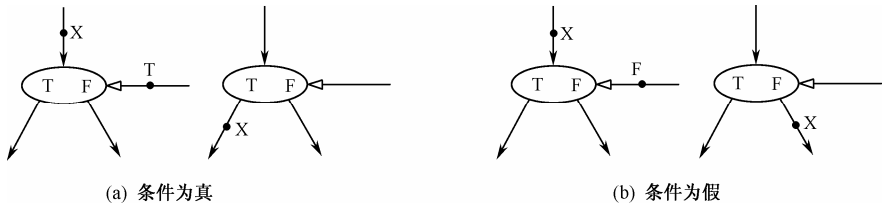


图 9-14 开关分支结点及其操作规则

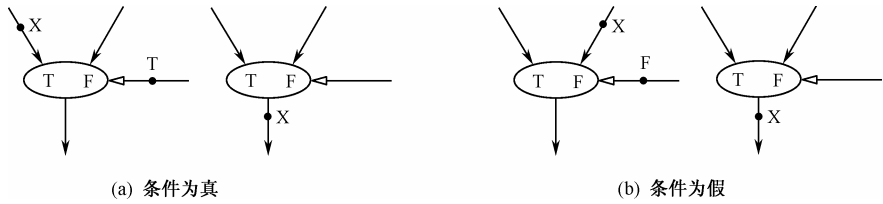


图 9-15 合并分支结点及其操作执行规则

5. 条件操作结点

条件操作结点用菱形表示，它有一条或数条数据令牌输入分支线和一条控制令牌输出分支线。条件操作结点根据数据令牌所带数值的某个特征（或若干输入数据令牌所带数值的某种关系）做出判断，最终在输出线上产生一个控制令牌。特征和关系通常包括： $X > 0$ ， $X = 0$ ， $X < 0$ ； $X > Y$ ， $X = Y$ ， $X < Y$ 等。图 9-16 所示是条件操作结点及操作规则示意图。图 9-16 (a) 所示是单输入分支线条件操作结点示例，图中 P 是判断真、假使用的条件；图 9-16 (b) 所示是多输入分支线条件操作结点示例。

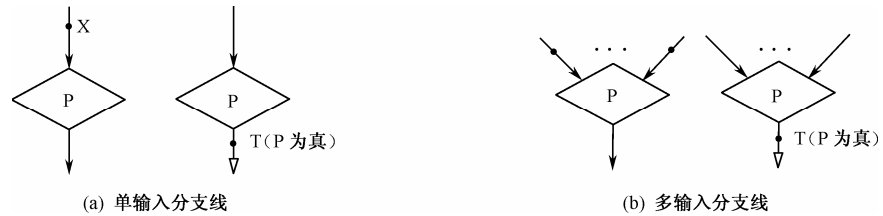


图 9-16 条件操作结点及其操作执行规则

图 9-17 所示是条件操作结点及其操作执行规则示例。图 9-17 (a) 所示是单输入分支线情况，当输入线上出现 X 且输出线上无数据令牌时，若 $X > 0$ 条件满足，输出线产生 T 控制令牌，否则，输出线上产生一个 F 控制令牌。图 9-17 (b) 所示是双输入分支线情况，其操作执行规则类同于单输入线，所不同的是判断条件改为 $X > Y$ 。



图 9-17 条件操作结点及其操作执行规则示例

根据数据流程图的需要，可以利用上述这几种单功能操作结点组合成功能复杂的多功能结点。下面举例说明这些单功能操作结点的使用方法。

【例 9-1】 利用上述单功能操作结点实现一般高级语言中的条件语句：

```
if true then G1 else G2
```

试画出数据流程图，其中 G1 和 G2 是各自独立的数据流程图。

如图 9-18 所示，利用一个复制结点、一个 T 门分支结点和一个 F 门分支结点实现起始数据令牌的两路传送，它根据起始控制令牌所携带的是真值还是假值，把起始数据令牌分别送往 G1 数据流程图或 G2 数据流程图，并利用一个合并条件分支结点选择 G1 或 G2 数据流程图中的一个结果作为输出，选择的依据仍然是起始控制令牌携带的是真值还是假值。

【例 9-2】 利用上述单功能操作结点实现一般高级语言中的循环语句：

```
while P do G
```

或

```
until P do G
```

试画出数据流程图，其中，P 是循环条件，G 是循环体。

如图 9-19 所示，为了使数据流程图中的循环体 G 能够开始执行，一开始要输入一个起始数据令牌和一个起始控制令牌，并用一个合并的分支结点取得循环体 G 的输入数据令牌。在第 1 次循环开始时从外部输入数据令牌，而在以后的各次循环中都从循环体本身的输出端取得所需要的输入数据令牌。在第 1 次循环时，由一个 T 门分支结点控制起始数据令牌是否输入给循环体 G，这就是 while 语句与 until 语句的区别。控制方法是起始控制令牌携带真值还是携带假值。另外，它用一个单输入分支条件操作结点依循环结束条件 P 产生的控制令牌来控制循环的执行。最后用一个条件分支结点分配每次循环产生的结果数据令牌。如果循环还没有结束，则条件操作结点 P 输出为真值，通过条件分支结点把结果数据令牌分配给循环体 G，继续进行下一次循环；如果循环结束，则条件操作结点 P 输出为假值，通过条件分支结点把结果数据令牌输出到外部。

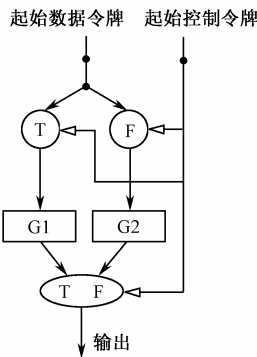


图 9-18 条件结构数据流程图

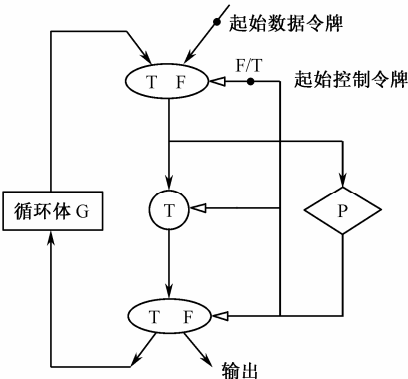


图 9-19 循环结构数据流程图

【例 9-3】 给定一个自然数 x ，求它的阶乘 $x!$ ，并画出数据流程图。

如果用 C 语言来实现 $x!$ ，则可以描述如下：

```
main()  
{
```

```
int x, i;  
scanf("x = %d", x);  
for (i = x-1; i>1; i-- )  
    x = x * i;  
printf("x != %d\n", x)  
}
```

图 9-20 所示是计算自然数 x 阶乘的数据流程图。从图中可以明显地看出，它有两个并行执行的迭代循环，一个是乘法操作循环，另一个是减“1”操作循环。为了使这个数据流程图能够开始执行，首先从外部输入一个带有原始数值 x 的数据令牌和一个带有假值的起始控制令牌。通过复制结点把从外部输入的原始数据令牌复制成完全相同的两个数据令牌分别送往乘法操作循环和减“1”操作循环；然后在起始控制令牌的控制下，两个迭代循环分别同时开始执行。每执行一次循环，“ >1 ”条件操作结点就产生一个带有真值的控制令牌，在这个控制令牌的控制下并行执行一次乘法循环和一次减“1”循环。当“-1”操作结点输出带有数值为“1”的数据令牌时，“ >1 ”条件操作就产生一个带有假值的控制令牌，在这个控制令牌的控制下停止乘法操作循环和减“1”操作循环，并把乘法操作结点产生的最后一个数据令牌作为最终运算结果输出到外部，在这个数据令牌中携带的数值就是最终运算结果 $x!$ 。

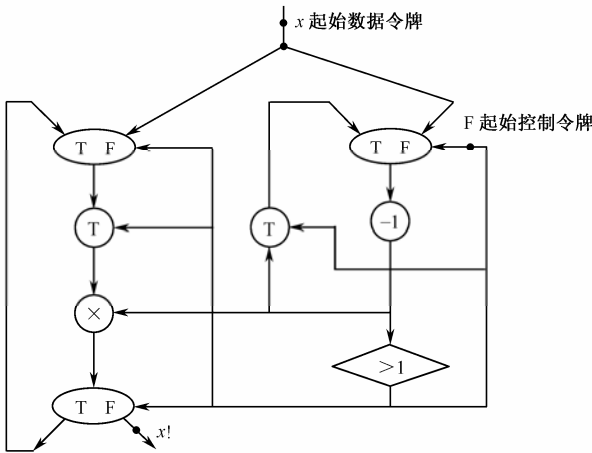


图 9-20 计算 $x!$ 的数据流程图

从图 9-20 所示的计算 $x!$ 的数据流程图可以明显地看到，在数据流计算机中操作执行过程的异步性和充分的并行性。

数据流程图相当于数据流计算机的机器语言。对于一般用户，如果直接用数据流程图编写程序很不方便。数据流程图与传统计算机的高级语言和超高级语言相比，不易被用户接受，因此必须研制适合于数据流计算机的高级语言，这种高级语言应该能够用类似于自然语言的方式最大限度地描述隐含的并行性，并具有易读、易理解和调试、维护方便等优点。

9.4.2 数据流语言及其性质

目前，数据流语言的研究还很不成熟，还没有形成像传统高级语言那样的规范版本。就

已经出现的数据流语言而言，大致可以分为如下三类。

(1) 单赋值语言 (Single Assignment Language)。该类语言中包括美国加州大学 Irvine 分校研制的 ID 语言 (Irvine Data Flow Language) 和美国 MIT 科学试验室的 Dennis 教授于 1979 年提出的 VAL 语言 (Value Algorithmic Language)。

(2) 函数类语言 (Functional Language)。该类语言中比较著名的有美国犹他大学研制的 FP 语言 (Functional Programming Language)。

(3) 命令类语言 (Command Language)。目前，美国依阿华州立大学正在研制此类语言，并研制了把命令类语言转换成数据流语言的编译器。

一般来讲，不论采用哪种高级语言编写程序，都可在使用相应的编译程序处理后，变换成数据流程图。但是，传统机器上广泛使用的面向过程的命令或语言缺乏并行性描述，很难表达数据流控制机制的并行性。虽然有些高级语言也扩充了并行描述部分，如 FORTRAN 的 FORK 和 JOIN，以及并行 PASCAL，Ada 等语言。由命令式语言程序转换成数据流程图的过程也还是复杂和低效的，所以仍需要研究和发展新的、适合于数据控制机制的高级语言。希望能以“自然”方式，最充分地表达计算的并行性，方便用户编程，并使程序有较强的可读性和可维护性。在这方面，美国的 VAL 语言、ID 语言，法国的 LAU 语言，英国曼彻斯特大学的 SISAL 语言等都具有可操作性。

MIT 所研制的 VAL 语言是面向数值的应用式数据流语言。用它书写的程序是一组独立翻译的模块集合，每一个模块包括一个外部函数定义，该函数对于程序中的其他模块都可以访问。模块还可以包括内部函数定义，它仅能在模块中使用，对其他模块则是不可访问的。总之，VAL 语言易于开发程序中隐含和显式的并行性，语句结构表达了算法的并行成分；没有变量的概念，仅有数值的名称，提供的数据类型有整型、实型、布尔型和字符型等；编写的源程序具有模块化结构，源程序中不规定语句的执行顺序，语句的执行顺序不影响最终计算结果；能对任何函数的自变量和计算结果的数据类型都在函数的首部加以定义。编译程序在编译过程中能方便地检查出函数和表达式中数据类型发生的错误。但是 VAL 语言缺乏输入、输出手段，程序的表达形式还不够自然和方便，运行的效率还很低。

传统的程序设计语言是建立在冯·诺依曼系统结构基础上的，共享存储单元，以顺序控制方式执行语句或指令。因此，控制流语言不能充分表达程序中的并行性。虽然已开发出并行 PASCAL，并发 PROLOG，向量 FORTRAN 等，但是这些语言是以顺序执行和多赋值为基础的，所以不能用到数据流计算机中。

数据流语言应具有如下性质：

(1) 并行性好。指令的执行顺序仅受程序中数据相关性的约束，而与指令在存储器中存放的位置（即地址）无关。因此，数据流语言能够以自然的方式最大限度地表达程序中的并行性。

(2) 单赋值规则。根据单赋值规则，在所有语句的左边，同一变量名只能出现一处，即要为任何一个重新赋值的变量选择一个从未用过的新名字，而且在以后所有引用中都使用这个新名字。例如下列两个程序段：

程序甲：	$x = a - b$	程序乙：	$x = a - b$
	$x = x + y$		$x1 = x + y$
	$z = x - y$		$z = x1 - y$

程序甲语句序列不满足单赋值规则，而程序乙却满足单赋值规则。单赋值规则使程序清晰，易于理解，为程序的并行执行提供了一种新方法。该规则由 Tesler 和 Enea 于 1969 年提出，被法国 LAU 机和英国的 Manchester 机采用（注：它们均为数据流计算机）。

(3) 不产生副作用。副作用是指使用了不当变量而产生的数据相关，从而导致程序不能顺序执行。在数据流语言中不使用全局变量和公共变量，严格控制变量使用范围，采用赋值调用 (Call by Value)，而不是传统机中的引用调用，这从根本上解决了变量的同义名问题。赋值调用过程只复制变量，而不修改变量。因此，在子程序中决不修改调用程序传送来的变量，即各程序模块之间的输入和输出是完全隔离的。

(4) 结果的局部性。传统的过程式语言大量使用局部变量，并只允许在本过程内部定义、赋值和引用，保证了操作结果具有局部性，不对其他过程产生影响。即使在不同过程中使用了相同名字的变量也仍然能够保证操作结果具有局部性。但是，过程式语言也允许使用全局变量，并规定任何一个过程都可对全局变量赋值，因此，全局赋值可能会对其他过程产生影响。在数据流计算机的指令中，没有长远起作用的数据依赖关系，数据流语言完全采用模块化结构，不使用全局变量，不允许全局赋值，对形式参量的赋值也有严格限制。因此，在数据流语言中每一个操作产生的结果都具有局部性。

(5) 循环程序迭代展开。从图 9-19 中可见，一个循环程序可以用一个环行结构的数据流程序图表示。数据流计算机本身就是一种环行结构，因此，数据流计算机对循环程序具有天然的适应性。但是，如果要把循环程序的不同次循环展开，进行并行计算，则必须在数据令牌中增加一种特殊的标记，这就引出了动态数据流计算的概念（见前述）。现在通过例子说明如何把循环程序展开进行并行计算。有一个用 C 语言编写的循环程序如下：

```
for ( i=1; i <= n; i++)
{
    x[i] = a[i] + b[i];
    y[i] = x[i] + c[i];
    z[i] = x[i] + y[i];
}
```

在每个循环体中存在三个“先写后读”的数据相关，因此每个循环体只能串行执行。按照常规计算方法，循环体部分只能用一个加法器运算，共需要 $3n$ 个机器周期才能执行完。

如果把相邻的三个循环体同时展开，并按照流水线原理重新进行调度，可以得到语义上完全等效的新循环程序。

```
x[1] = a[1] + b[1];
y[1] = x[1] + c[1];
x[2] = a[2] + b[2];
for ( i=1; i <= n-1; i++)
{
    z[i] = x[i] + y[i];
    y[i+1] = x[i+1] + c[i+1];
    x[i+2] = a[i+2] + b[i+2];
}
```

$$z[n-1] = x[n-1] + y[n-1]; \quad y[n] = x[n] + c[n];$$
$$z[n] = x[n] + y[n];$$

新循环程序中，循环体内不再有任何数据相关，因此可用三个加法器并行执行。循环程序的前面部分称为装入部分，后面部分称为排空部分，同一行语句之间无数据相关，可以并行执行。所以执行新循环程序只需用 $n+2$ 个机器周期，运算速度提高 3 倍。

上述方法与硬件流水线技术相似，因此也称软件流水（Software Pipelining）技术。使用该技术可把循环程序的多个循环体重叠起来并使用多个处理机并行执行。

9.5 数据流计算机的评价

9.5.1 数据流计算机的优缺点

数据流计算机在许多方面的性能优于冯·诺依曼型计算机，现将已经模拟实验验证的主要优点和缺点归纳如下。

1. 优点

（1）高并行性。把程序转化为数据流程图，较好地开拓了其并行性，它不仅取得构造规格化，而且可取得很大的并行性。

（2）由于在指令中直接使用数值本身，而不是使用存放数值的地址，从而能实现无副作用的纯函数型程序设计方法，可以在过程级和指令级充分开发异步并行，可以把实际串行的问题用简单的办法展开成并行问题计算。

（3）与 VLSI 技术匹配。利用 VLSI 技术，可以容易地实现数据流计算机中许多模块单元的规则制造。

（4）提高了程序设计的生产效率。由于采用函数程序设计语言，自动向量识别的能力大大提高，对数据流的分析、处理就很容易了。比起控制流计算机使用的 FORTRAN 等语言，可提高程序设计的生产率。

2. 缺点

（1）与冯·诺依曼型计算机相比，数据流计算机的各种开销要大得多，使得并行中所获得的好处在实际应用领域中得不到体现。例如，指令级数据驱动引起每条指令额外流水线开销，使系统性能降低。

（2）数据流程序占据过多的存储空间，且容易引起存储器访问的冲突。

（3）当指令单元和处理单元数目增加，使系统规模扩大时，系统的瓶颈将是通信机构，包括开关网络剧增，这会严重影响系统的成本。

（4）数据流计算机完全放弃了传统的结构，独树一帜，不能吸收传统计算机研究领域已经证明行之有效的许多成果。值得提出的是，数据流语言尚不完善，用其描述隐含的并行性，使程序的调试工作变得非常困难，目前还缺少有效的解决方法。

9.5.2 数据流计算机需解决的问题

从上述对数据流计算机优缺点的论述可见，数据流计算机设计中尚需解决下列主要技术问题：

- (1) 研制易于使用、易于用硬件实现的高级数据流语言。
- (2) 研究程序分解、分配给各个处理部件的算法。
- (3) 设计出性能价格比高的信息包交换网络，实现资源冲突的仲裁和数据令牌的分配等大量通信工作。
- (4) 对于静态和动态数据流计算机，都要研制智能化的数据驱动机构。
- (5) 研究如何在数据流环境中高效率地处理复杂数据结构。
- (6) 研制支持数据流运算的存储系统和存储分配方案。
- (7) 在广泛的应用领域内，对数据流计算机硬件和软件进行性能评测。
- (8) 研究数据流计算机的操作系统。
- (9) 开发数据流语言的跟踪调试工具。

第 10 章 软件对系统结构的影响

计算机系统是硬件和软件有机地结合在一起而组成的，它们相辅相成，缺一不可。现代计算机的发展，使软、硬件之间的相互依赖、相互支持、相互渗透更为明显。本章提纲挈领地叙述操作系统、语言发展、并行处理、软件固化与硬化等对系统结构的影响，以及有关的基本概念。

10.1 操作系统的影响

现代计算机系统是一个相当复杂的系统，由丰富的硬件资源和软件资源组成。硬件资源主要包括中央处理机、内存储器、输入/输出设备和外存储器。软件资源通常是指语言处理系统、数据库管理系统和操作系统等。操作系统是必不可少的、与硬件关系最为密切的系统软件，它是控制和管理计算机系统资源、方便用户使用的程序的集合。操作系统的用途主要有两个：一是方便用户使用，它是用户与裸机之间的界面；二是提高资源利用率，管理好资源的分配和回收，合理地组织计算机系统的工作流程，使各种资源能协调有效地工作，以完成各种应用任务。所以，系统结构为操作系统奠定了物理基础，而操作系统使系统结构各部件的潜能得到充分的发挥。系统结构的某些功能可由操作系统完成，而操作系统的某些管理功能可由系统结构体现。操作系统的基本功能可归纳为处理机管理、内存管理、外部设备管理和信息管理。

中央处理机是计算机系统中最昂贵的硬件资源，为了提高它的利用率采用了多道程序设计技术。处理机管理功能负责记录各个进程的状态，按一定的策略把处理机分配给某个或某些（在多处理机的情况下）进程和从运行进程处收回处理机，管理进程状态的变迁。处理机管理也称进程管理。在进程控制块的存储结构、进程优先级的处理、进程调度算法的实现等方面，均可对系统结构提出需支持的要求。而硬件的支撑将使进程管理的效率明显提高。

存储管理功能是用适当的数据结构记录系统中主存储器的使用情况，按一定的策略在多道程序之间分配主存空间，保护主存储器中的信息不被别的程序有意、无意地破坏或使用。系统结构能给予存储管理越来越多的支持，存储系统的层次结构、替换算法的实现、存储保护的结构、地址交换的实现等属于系统结构解决的问题，伴随这些问题的解决使存储管理的方式也随之改进。各种存储管理方案均以存储分配方式为基础，配以相应的地址转换机构和保护机构，故不同的存储分配方式将影响存储的系统结构。

外部设备管理功能包括记住系统中各类设备的使用状态，按设备的特点用适当的策略将设备分配给各道程序使用，并提供启动设备和处理它们的中断服务。通常把外部设备及其接口电路、控制部件和管理软件统称为 I/O 系统，其中的管理软件就是操作系统的设备管理部分，其余是系统结构范围内的硬件。真正的 I/O 操作，可把用户从接口、控制器以及设备等繁琐的物理特性中解脱出来。用户只需掌握操作系统提供的命令、语句和系统调用的使用方法就可以“随心所欲”地调用外部设备，不再需要用户自行编制具体的、涉及硬件特性方面

的程序，从而达到方便用户使用的目的。为了克服外部设备与主机在速度上不匹配的缺点，使主机和外设并行工作，提高主机和外设的资源利用率，操作系统普遍使用了中断、通道、缓冲区等技术。这就要求计算机系统具备完善的中断系统和通道结构。

信息管理通常被称为文件系统，其功能主要涉及文件本身的逻辑组织、存放在外存储器上的物理组织、外存空间的分配、目录结构以及对文件的操作。为了便于对信息进行管理，所有在外存中的信息均以文件的形式存放。文件系统负责文件的存储、存取、修改、转储和保护，为用户提供一个按需存放的良好服务环境，使用户完全从外存的物理特性、文件在外存的物理分布等细节中解脱出来。文件系统还提供各种保护措施，防止由于各种偶然或者人为事故造成对文件的损坏或泄密。因此，计算机系统结构应配置相应的外存储器层次，有良好的中断、DMA 或通道等结构，有硬件的保护措施（如计算机病毒检测卡、消毒卡等）。

当今计算机系统的应用领域如此广阔，从科学计算到工业控制，从办公自动化到软件开发，等等。因此，除了管理系统资源外，还为适应不同应用要求，出现了各具特色的性能指标各异的操作系统。与之相对应，计算机的系统结构也出现了侧重点。

10.1.1 批量处理系统

批量处理系统应用于大型科学计算。用户把计算程序和有关的数据一起交给计算机系统，由输入设备将它们输入到主存，计算机花费大量的时间进行运算，计算结果从输出设备输出。每个用户提出的一个计算任务称为一个作业，非会话型作业可以脱机进行。早期批量处理系统只支持单道程序运行，即每次只为一个作业服务。而现代的批量处理系统，由于磁盘和通道的普遍使用，作业可以随时（不必集中成批）进入系统，存放在磁盘输入井内（磁盘上一个专门的存储区域），形成作业队列，操作系统将按一定的策略从作业队列中取出一个或多个作业进入主存运行。计算结果也可先存放于磁盘输出井内，待输出设备可用时再输出。因此，“批”的概念已不十分明显。多道批处理系统要求结构全面，硬件设备齐全，有较完整的存储层次结构，重视资源的利用率，作业的吞吐量是主要的性能指标（所谓“吞吐量”是指在给定时间内，一台计算机所能完成的总工作量）。

10.1.2 单用户交互式系统

单用户交互式系统用于软件开发和办公室自动化。用户与系统之间需要联机通信，交互地进行工作。用户将命令发给操作系统或正在运行的程序，并且能立即收到它们的响应或输出结果。由于系统是与进行交互，所以系统的主要性能指标是响应时间（即从用户发出命令到开始看到输出结果的时间间隔），它应保证在人可以容忍的等待时间范围内。单用户交互式系统的主机为个人计算机，其输入设备为键盘、光笔或鼠标器，输出设备为打印机、显示屏或绘图仪。它的存储层次较简单，往往只有主存和外存（常用磁盘）。它有完善的中断系统，以缩短响应时间。在外存与主存交换大批量信息时常使用 DMA 方式。

10.1.3 分时操作系统

将中央处理机的时间轮流分给各个联机用户的工作方式称为分时方式，具有分时方式的操作系统称为分时操作系统，它用于多用户交互式系统。为了避免处理机的时间被某个用户或进程独占，操作系统将时间分成很小的一段（如 50ms），称为时间片，每个用户或进程依

次得到一个时间片。如果该时间片未用完，系统已完成该用户或进程所要求的任务，则系统提前转向下一个用户或进程；如果在该时间片内未完成该用户或进程所要求的任务，系统暂停为其服务，转向其他用户或进程，等下一轮再为其继续服务。这种系统的主要性能指标与单用户交互式系统一样，也是响应时间。采用轮转法分时方式，可以保证各个用户对响应时间的要求，使他们好像独占计算机，并且能交互式工作。这种系统的主机往上是小型机、大型机等，主机速度快，主存容量大，存储系统的层次结构较丰富，一般均有 Cache。系统还具备定时部件、完善的中断系统和通道结构。

10.1.4 实时操作系统

当计算机系统应用于过程控制和信息处理领域时，均有一定的实时要求。所谓“实时”表示立即、现在的意思，是指计算机系统能及时响应外部事件的请求，并以足够快的速度完成对事件的处理。实时操作系统可分为下列两类。

1. 实时控制系统

过程控制一般是指一个计算机系统对一个工业生产过程进行监控，也可以扩大到环境控制或参数监测等。这类应用的共性是反馈，即计算机从被控制过程得到输入，然后计算出一个结果，启动相应机构做出响应，以保持过程的稳定性。为了保持系统处于平衡状态，应在严格的时间范围内做出响应，尤其是威胁到系统安全时更应如此。这个时间约束值随被监控的对象而异，可能为秒级、毫秒级、甚至微秒级。实时控制的另一个特点是可靠性要求高，在任何硬件发生故障时，要保证系统仍是安全的。因此，实时控制系统要求具有非常完善的、响应极快的中断系统；有丰富的、适应各种要求的 I/O 接口（如并行的、串行的、A/D 和 D/A 等接口），有调整能力很高的、不间断供电的电源；有坚固的抗震性能好的机箱；有很好的正压通风系统，可抗粉尘；使用高等级芯片，可耐高温；关键硬件部件有相当冗余，以提高系统的容错能力。

2. 实时信息处理系统

实时信息处理系统接收来自终端的服务请求，在短时间内（如数秒）对用户做出正确回答。事务处理的实时性还在于使数据库及时更新，以包含最新信息。典型应用是机票预定、银行事务处理、资料查询、军事指挥等方面。所以，实时信息处理系统的结构要求具有很完整的存储层次，有响应极快的中断系统，有完善的通道结构，有很好的容错能力。

10.1.5 网络操作系统

网络中各台计算机配有各自的操作系统，网络操作系统把它们有机地联系起来。其功能主要是提供各台计算机之间的通信和实现网络资源共享。网络通信模块是网络操作系统和普通操作系统的主要区别所在，它是本地操作系统与网络之间的接口模块，使本地系统中的进程能方便、有效地使用网络中的各种资源，并且为网络中其他用户使用本地资源提供服务。所以，网络中各台计算机的系统结构内必须配置通信接口部件，以实现网络通信协议物理层和数据链路层的功能。

10.1.6 分布式操作系统

分布式系统是不共享存储器或时钟的一群处理机。处理机之间通过各种通信线路相互通

信, 以达到资源共享、提高计算速度(分布式并行计算)、提高可靠性(某个处理机故障时不影响整个系统正常运行)等目的。分布式系统中的处理机在容量和功能上不尽相同, 它们可以是微机、工作站、小型计算机或大型通用计算机系统。它们都有自己的局部存储器。分布范围大的可利用电话线通信, 范围小的可利用高速总线、同轴电缆、光缆、无线信道等传输介质进行通信。

分布式操作系统有下列 4 个特点。

(1) 进程之间通信没有公共存储器可用, 需要通过传输介质传送消息。

(2) 系统资源分配要考虑它们分布在多个场地的特点。

(3) 各个处理机既能自主地管理分布于本场地的资源, 又能接受和处理其他场地对这些资源的请求, 即系统资源的操作是高度自治的。

(4) 不失时机地协调系统中各场地的负载, 使之基本均衡; 充分发挥各场地的作用, 提高整个系统的效率。系统的协调、平衡对用户是完全“透明的”。

分布式系统与计算机网络的区别主要在于: 前者是一个完整的一体化的系统, 用标准、统一的界面(如系统调用命令和数据格式)去使用系统资源, 实现各种操作; 后者要靠网络通信模块实现用户命令/响应与协议命令/响应之间的翻译与转换工作, 而且分布在各个处理机上的合作进程难以实现同步协调。

从分布式操作系统的特点出发, 要求分布式系统的系统结构具有下列特征:

(1) 把多个计算机模块通过信息传输系统(Message Transfer System, MTS)连接起来, 可采用简单的分时总线及环状或星状等拓扑结构。

(2) 每个计算机模块, 除 CPU 外, 还应有局部总线、局部存储器、I/O 接口、外部设备, 以及 MTS 接口部件, 如通道和仲裁开关(Channel and Arbitrate Switch, CAS)。

(3) 在通道和仲裁开关内有高速通信存储器, 用于缓冲传输的信息块。通信存储器给每个处理机提供通信的逻辑端口。不同处理机任务之间的通信, 都是经过通信存储器实现的, 而同一个处理机任务之间的通信, 只通过局部存储器完成。

(4) 灵活多样的结构可实现各种复杂的机间互连模式, 从而适应多样的算法, 扩大并行处理的范围。

(5) 采取特殊的同步措施, 保证程序所要求的正确顺序, 实现在指令、任务、作业级的并行。

10.2 语言发展的影响

在第 1 章, 我们已分析过计算机系统的层次结构, 最低层是硬件构成的机器, 而其上的各层机器是由机器指令、操作系统、汇编语言、高级语言、应用语言等构成的虚拟机器。随着程序设计语言和编译技术的发展, 对系统结构的影响也逐步加深。在本节中, 将讲述多层计算机的有关技术和程序移植、仿真等概念。

10.2.1 实现新层次的方法

实现新层次有三种不同的方法, 它们是解释、翻译和程序扩展。

1. 解释

从本质上讲，解释就是读取、分析并执行处在某个程序里的指令的一个进程。正在被解释的那个程序就是供解释程序使用的数据，执行解释的那个进程所起的作用和一个“硬件”处理机完全一样，它们的速度和所经过的状态序列也完全一样。因此，解释程序构成计算机系统层次结构中的一个层次。其中，每一个解释程序都解释在它上面的那一层。

2. 翻译

翻译为有效地实现一个复杂语言层次提供了一种方法。翻译程序先读入并分析源程序，然后生成一个新的程序（目标程序）作为它的输出。目标程序是由另一种不同的语言（目标语言）组成的。对翻译而言，主要考虑的是如何尽力避免重复执行浪费时间的语法分析。所以，目标语言要具有语法简单的特点。汇编程序、编译程序、宏处理程序是翻译程序中最主要的三种。汇编程序所翻译的对象是符号化的指令，即用机器码来置换表示变量和符号的操作码，形成可执行的目标程序。编译程序是把面向问题或面向对象的语言翻译成便于解释的一种形式。编译和汇编的差别主要是翻译程序上的不同。如果翻译过程中所要执行的语法分析量很大，这个翻译程序就被称为编译程序，否则就被称为汇编程序。宏处理程序是先读入并保存一系列宏指令定义，然后再读入被翻译的程序，找到宏指令的调用并用在这之前已定义的正文来置换它，形成一个文件，供汇编或编译时使用（即翻译的对象）。往往将汇编与宏处理结合在一起，构成宏汇编程序。

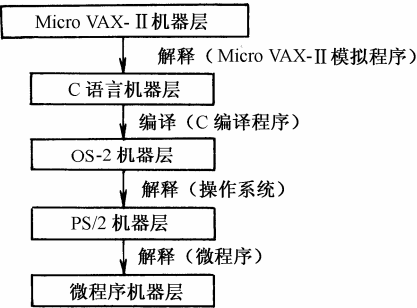


图 10-1 用模拟方式构成多层计算机

利用语言的解释和翻译功能，就能构成多层计算机。图 10-1 所示是用 C 语言所写的一个 VAX-II 的模拟程序，把 PS/2 视作一个 5 层的机器。

3. 程序扩展

所谓程序扩展就是用多种语言写成的一个批过程。对用户而言，将代表一个过程的语句视作一条“指令”，用这样的“指令”组合成各种程序，类同于用汇编语言编程。从某种意义上来说，这些“指令”确定了一台虚拟机，可以通过翻译或解释实现这台虚拟机。这种使用一些过程来构成一个层次的方法不及前两种广泛。

10.2.2 多层计算机的设计策略

设计一个多层计算机系统有三种在概念上不同的方法。

1. 从上往下的设计

从直观上看，从上往下设计的基本原理是很明显的。设计人员选用这种方法时，要极认真地确定供用户直接使用的那层（即最高层）虚拟机器的语言，以及在这一层上的数据类型和对应的指令。在顶层被完全而精心地确定下来之后，根据顶层的执行过程再考虑下一个层次，即为顶层写一个编译程序或解释程序。然后，依次类推，直至物理机器层为止。如图 10-2 所示为一个图书馆管理设计的多层计算机系统，A 层是用户使用的虚拟的图书馆管理计算机系统，在这层上具有登录、检索、统计、增加、删除、打印等功能。B 层是为实现 A 层而选用的（或创造的）某种程序设计语言。针对图书馆管理的需求，该语言应能方便地处理字符串。C 层则是为 B 层提供一个运行的软件环境，如某个操作系统。D 层是物理机器，它应当

具有一个容量足够大的存储系统，以容纳图书馆管理所需的所有信息，同时，应具有相当的字符处理能力。在购置或设计 D 层机器时，如能对整个系统的性能进行评估，将是十分有益的。设计一个模拟程序，这就是 E 层，通过 E 层的模拟，能对系统结构进行部分测试并进行评价，以最终确定 D 层是否可取。当然，E 层的模拟依托于某个计算机系统，所以 E 层以下还有若干层是用来支持它的，在图 10-2 中没有画出。

2. 从下往上的设计

从下往上的设计是基于现成的硬件（物理机器）开始进行的设计。在硬件上构造出一层，它的性能要比硬件更便于应用。新构造出来的这一层随后成为能创造出更为方便的虚拟机器的一个基础。在实现最终的用户虚拟机器之前，在这一层之上需构造若干层次。图 10-3 所示为从下往上设计的多层操作系统。它是从传统机器层开始的，而不是从微程序机器层开始的，这就是 A 层。传统机器（即物理机器）都有中断系统，故程序执行时，精确的状态时序取决于中断出现的时间，而外部设备的请求又是随机的。因此，一个程序重复运行若干次，每一次运行所表现的状态时序不可能完全相同。一个进程，当它的运行状态不能完全由它的程序、数据以及它的起始状态所确定时，这个进程被称为不确定进程。由于在一台确定的机器上进行程序设计比在不确定的机器上容易实现，所以在物理机器层上面为确定进程的机器，即 B 层。B 层具有若干协同操作的“并行”运行着的确定进程，而不是只具有唯一的不确定进程。为了扩大用户使用的存储空间，在 B 层上增加了虚拟存储器，即 C 层。C 层的特性是：所有的进程都是确定的；进程可使用虚拟存储器，而虚拟存储器对进程没有干扰。D 层是虚拟 I/O 机器，提供虚拟的 I/O 指令。虚拟 I/O 机器上的一条指令，由下一层的一个进程来实现。

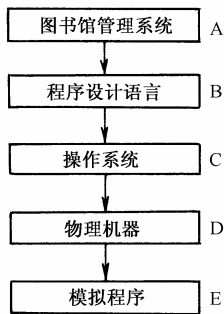


图 10-2 从上往下设计的“图书馆管理”计算机系统

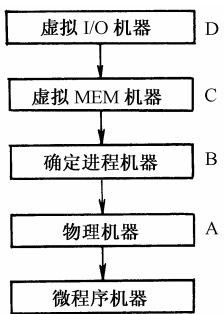


图 10-3 从下往上设计的多层操作系统

3. 从中间开始的设计

实际上很少有完全是从上往下或从下往上进行的设计。从上往下设计的缺点是最上层虚拟机的某些特性在硬件上实现很困难，从下往上设计的缺点是设计的层次可能相当多。从中间开始设计，也就是先把物理机器层确定下来，然后，再往上、往下展开。往上是设计操作系统和汇编程序、解释程序、编译程序，往下是设计微程序，能同时在两个方向上进行。

10.2.3 程序移植

为了增加程序的可移植性，近年来已研究出了多种方法，现简述如下。

1. 通用程序设计语言

设计一种通用的语言，所有程序员均用这种语言进行程序设计。在各种类型的计算机上，

建立一层由该语言所定义的虚拟机器（即为所有计算机提供该语言的编译程序或解释程序），就可以很方便地把一个程序从一台机器移植到另一台机器，如图 10-4 所示。

但是，通用程序设计语言至今未获成功，其根本原因是还未能满足如此众多的应用要求。何况，用某种语言编制的程序在这台机器上能运行，在另一台机器上就不一定能运行。这主要是受机器的数据格式、字长、存储系统、操作系统、指令系统等因素的影响。所以通用程序设计语言对于程序移植固然很好，但很难实现。

2. 强力逼近

为每一台机器上的每一种语言都设计一个编译程序或解释程序，程序员不管选用何种语言，总可得到一个有效的翻译程序，在由那个语言所定义的虚拟机器上能够运行。如现在有 L 种语言，有 M 种机器，就要设计 LM 个翻译程序，如图 10-5 所示。由于新的语言和新的机种不断涌现，需编制的翻译程序不胜其多，因此本方式也是无法真正实现的。

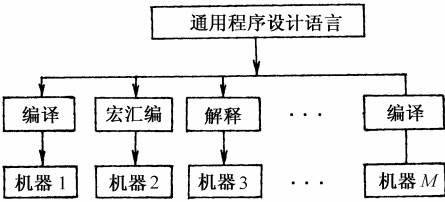


图 10-4 通用程序设计语言的层次结构

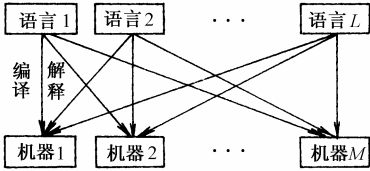


图 10-5 强力逼近的层次结构

3. 通用计算机语言（UNCOL）

设计一台 UNCOL 所确定的虚拟机，并用这台虚拟机实现各种语言层，如图 10-6 所示。在 UNCOL 层基础上，为各种语言编写编译程序或解释程序。为了利用 UNCOL 所编写出来的编译程序或解释程序，就需在每一台计算机上实现 UNCOL 层次，即为每台机器配置一个能把 UNCOL 翻译成机器语言的翻译程序。使用 UNCOL，只需 $L+M$ 个翻译程序，比强力逼近的 LM 个翻译程序少得多。而且，不同型号的机器都有一个相同的 UNCOL 层，即使这些机器在硬件方面相差很大，但它们的程序却可很方便地互相移植。UNCOL 是个实用的方法，IBM 公司在 360 系列和 370 系列机上采用该方法，获得巨大利益。但使用 UNCOL 方法后，程序的执行速度受到了影响。

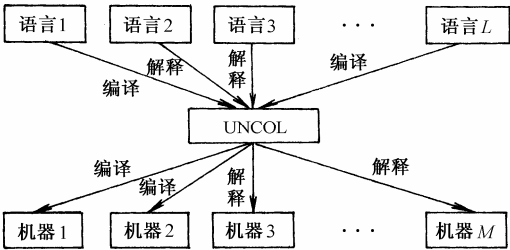


图 10-6 使用 UNCOL 在 M 台计算机上实现 L 种语言

4. 自虚拟机

用 UNCOL 方法可以解决同一系列不同型号机器间的程序移植，却不能解决不同厂商所生产的不同型号机器间的程序移植。为此，可采用自行设计虚拟机的方法来解决，使可移植

的软件运行在虚拟机上。虚拟机必须遵循下列准则：必须适合于可移植的软件；必须很容易地在现行机器上实现。这类虚拟机最常用的方法是宏指令的展开，如图 10-7 所示。把虚拟机上的每一条指令的执行过程都写成一条能扩展成现行机器的汇编语言的宏指令，将这些宏指令展开后，就得到了用现行机器的汇编语言所写成的一个程序，即可实现程序移植。虚拟机的宏指令处理程序，可以是宏汇编，或一个通用的宏指令处理程序。通用宏指令处理程序是功能很强的程序设计语言，能处理各种类型的输入语言，使移植软件更灵活和方便。但是，在使用它之前，它本身必须在各种目标机上执行。为此，需将通用宏指令处理程序进行预处理，得到它能在目标机上可执行的文件，如图 10-8 所示。

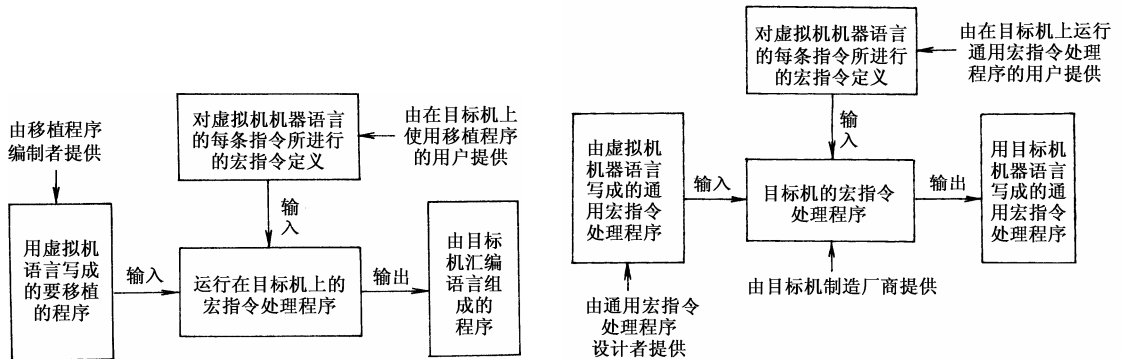


图 10-7 使用虚拟机形成可移植软件

图 10-8 由目标机获得通用宏指令处理程序的过程

5. 仿真

在第 1 章里已提出仿真的定义：用微程序直接解释另一种机器的指令系统的方法。使用仿真程序，便可将目的机的程序移植于宿主机。但是，微程序机器级的系统结构是按优化实现传统机器级的系统结构来设计的，一旦设计并制成后，在同一个微程序机器级的系统结构上难以优化仿真差别大的另一种传统机器级的系统结构。因为微程序级深深依赖于传统机器级的系统结构，故宿主机与被仿真的目的机系统结构不能差别过大。要推动仿真工作开展，一方面要开展通用宿主机结构的研究，拓宽优化仿真的范围。例如，位片式微处理器就是通用宿主机的一种，它可配上不同的微程序以构成多种型号机器。另一方面要开展微程序的软件研究，只有微程序汇编语言及对应的模拟程序是不够的，还需要有微程序高级语言和微程序开发系统，以便使用软件工程的手段促进其发展。

6. 网络

利用异种机连网的优势，将程序通过网络传输到能执行的机器上运行，其结果再由网络传回来。这种方法效率高，不需要上述的种种预处理，也不存在同步问题，为程序移植提供了一个好的解决方法。但网络投资大，时间开销也较大。

10.2.4 现代模型的研究方法——计算机仿真

在计算机上建立系统模型，并运行和研究这个模型，是现代模型研究方法在方法学和工具上的革命性变革。计算机被称为“活的数学模型”。计算机仿真经历了模拟计算机仿真、混合计算机仿真、全数字计算机仿真三个时期。当前全数字仿真技术的发展促进了仿真方法学、仿真算法、并行仿真技术、仿真软件以及人工智能技术的运用和发展，进而把计算机仿真的

应用领域从传统工程领域扩展到社会、经济、生物等非工程领域。当前，全数字仿真具有下列 10 个特点。

(1) 全数字实时仿真技术有所突破。集中参数复杂系统的全数字实时仿真已经成为实时仿真的主流。研制出了基于全并行处理原理、采用 MIMD 及变字长结构的实时全数字仿真计算机。

(2) 高性能仿真工作站的出现。工作站以 32 位微处理机为基础，集交互式图形、仿真软件及计算机网络于一体，为用户提供一个友好的信息仿真工作环境。

(3) 小巨型机与分布参数系统仿真。由于 VLSI 技术的发展，采用多个相同的标准处理单元进行并行处理有了新的进展，出现了小巨型机。这种系统有多个处理单元 (PE)，数量可由用户选定，计算速度随 PE 的数量增多而增长，在向量处理方面有很强的能力。在用于分布参数系统仿真方面，具有很高的性能价格比，超过了传统的巨型机。

(4) 先进的仿真方法学及一体化仿真软件。仿真的基本概念框架、结构化建模及实验框架为先进的仿真方法学奠定了基础。仿真软件由此开始了一个新的时期。它在结构上具有显著特点：建模、实验、分析的概念框架；模型的分解与合并的结构化建模；实验框架的完整性等。多功能、多用户一体化仿真软件以数据库为核心，实现了模型数据、仿真运行数据、仿真结果的输出数据、输入及输出显示参数值等资源的共享，并实现了建模、仿真、分析、结果处理的一体化。

(5) 人工智能技术 (AI) 在仿真中的应用。AI 与仿真在学科上交叉涉及三个主要方面。

① 知识库用于建模与仿真研究，在仿真研究的各个不同阶段采用人工智能技术辅助工作；

② AI 技术与仿真技术的结合，在复杂大系统的仿真中建立基于知识库的专家系统模型；

③ 仿真模型的知识表达，利用 AI 技术建立面向目标的仿真语言，从而进一步改善建模与仿真过程中的人机交互环境，实现仿真程序的自动生成和自动检验。

(6) 海量并行处理的发展。海量并行处理通常指系统中处理单元的数量超过 1000 个，目前已达到 64×10^3 个，并相应开发了新的并行算法及软件。

(7) 连接式并行计算机体系结构。目前多数的多处理机并行处理系统采用 MIMD 系统结构，连接式计算机却采用了 SIMD 系统结构，不过在处理单元的网络连接方面有很大不同。它将 64×10^3 个处理单元划分为 4 个分区，各分区通过 4×4 交叉连接器与多个前端处理机相连。每个处理单元包括一个 ALU 及 64kb (8KB) 的 RAM。每个处理机与相邻的 4 个处理机相连，16 个处理机组成一个集成部件，部件间的连接采取十维的布尔超立方结构。连接式计算机具有先进的计算能力和灵活性，使分布参数系统的仿真获得了更强有力的手段。

(8) 神经网络计算机。神经网络的研究对计算机仿真技术的发展带来了革命性的影响，混合技术有可能再次成为发展趋势。

(9) 仿真操作系统和仿真平台。仿真操作系统具有对模型、参数集、实验框架、算法及实验结果等资源进行管理和调度，并提供友好的人机界面等功能。仿真平台是指仿真的支援环境。以此为基础，基于先进的仿真方法学发展仿真软件，将使仿真软件向一体化、规范化、标准化、智能化方向发展。

(10) 专家仿真系统的研究。计算机仿真是一种数值求解的方法，它通过系统的近似模型在计算机上获得数值解，而不是获得通解。为了使解优化，必须重复迭代求解。由于迭代优化的非结构化，因此不一定能得到最优解。为了减少求解次数和获得最优解，引入专家经验

可取得决定性的成果。基于这种思想,提出了把专家系统接入仿真回路,利用专家系统辅助重复迭代仿真过程,实现解的优化。与已有的以咨询为主的仿真专家系统比较,反馈式专家仿真系统的实现,将使专家系统纳入仿真实验回路中并取得了显著的成效。

10.3 并行处理的影响

提高计算机系统的性能可从两种途径入手:一是提高器件开关速度;二是改进计算机系统结构的处理技术。器件技术的发展是推动计算机发展的主要动力,也是最活跃的因素,突出地表现在集成度、速度和可靠性等方面不断提高。但单纯地依靠微电子技术来提高处理机速度受到两方面阻碍:一是信息传输速度受光速的限制;二是物理尺寸受氢原子直径的限制。因此,从改进系统结构着手是提高处理机速度的重要方向。并行处理技术是改进计算机系统结构的关键技术,有着巨大的潜能。

并行处理是指开发计算过程中并发事件的一种有效的信息处理形式。并发性包括并行、同时及流水。因此,并行处理存在于多处理机系统、单机系统以及单机系统的一些功能部件内。

10.3.1 并行计算机分类

(1) 流水线计算机。它是在功能单元内实现重叠操作来开发时间并行性。

(2) 阵列机。它采用多个同步处理单元(PE)来实现空间并行性。各个 PE 可并行地执行相同的操作。

(3) 多机系统。它通过多个处理机共享资源(如存储器、数据库等)来获得异步并行性。

并行计算机有两种基本形式:SIMD(单指令流多数据流)和 MIMD(多指令流多数据流)。流水线机和阵列机是对一条指令及其有关的多个数据进行处理,属于 SIMD 类;而多处理机系统可以同时多条指令及其相关的数据进行处理,属于 MIMD 类。

10.3.2 并行计算机性能

使用并行计算机的主要目的是提高程序执行的速度。当一台有 N 个处理机的并行机以峰值性能运行时,它的处理速度应是单机的 N 倍。但是,并行机的持续性能很难一直维持峰值状态,主要原因是:处理机间通信带来的开销;进程之间的同步开销;任务划分不均匀而使一些处理机空闲所导致的效率下降;控制系统及调度任务所带来的开销;互连网络的带宽及结构的影响;数据存取模式的影响等。

并行机系统的性能通常是以其速度衡量的,常用指标是加速比 S_p 和效率(或使用率) U 。如果一个程序在单机上的执行时间是 T_1 ,而在有 N 个处理机的并行机上执行的时间为 T_N ,则加速比为 $S_p = T_1/T_N$,效率为 $U = S_p/N$ 。显然, $0 < S_p \leq N$, $0 < U \leq 1$ 。由于并行机一般达不到峰值状态,故 $S_p < N$, $U < 1$ 。

10.3.3 并行处理技术中的基本问题

1. 算法问题

并行算法与并行机的关系远比串行算法与串行机的关系密切。串行算法的优劣与具体机

器的结构之间无多大联系，在一台串行机上是最优的算法在另一台串行机上也往往是最优算法。而在并行机中则不然，算法优劣直接依赖于并行机的系统结构。最佳串行算法与最佳并行算法之间也没有必然的联系。用并行算法描述问题的并行性需要并行程序设计语言和支撑并行处理的操作系统的支持。设计并行算法是很困难的。首先，在处理机很多的情况下，要把任何一个问题分割成足够多的并行进程就不容易，而且也不是所有问题都能做到。其次，设计算法时还要考虑处理机之间的通信开销，使其尽可能小。目前，SIMD 的同步算法已经基本成熟，而 SIMD 异步算法和 MIMD 算法还有待进一步研发，其稳定性和误差的传播特性以及收敛速度和收敛条件等还有待于研究。

2. 软件问题

软件问题主要是指程序设计语言、操作系统、编译程序和利用整个并行处理能力的程序设计问题，以及这些软件工具与现有软件工具的接口问题。它是目前影响并行机性能的主要障碍。因为软件好坏对性能高低的影响可相差 50~100 倍，甚至可能高达几个数量级。

在多机系统中，提高程序运行效率的关键是既要程序分解为足够多的进程，又要尽量减少它们之间的同步与通信开销。因此，要求并行程序设计语言具有明确表达和抽取进程并行性的语句，以便在程序运行时提供相应的控制和管理手段，如并发任务的派生、通信和调度等。

解决并行程序设计语言问题有两个途径：① 将传统串行语言原封不动地移植到并行机上，由并行编译程序完成程序的并行化，或者在传统串行语言中增加并行控制语句，借助编译程序完成程序的并行化；② 设计新的程序设计语言。开发具有自动识别串行程序并行性能的编译程序是解决这一问题的可能途径，尤其是在现行语言中增加并行控制语句的方法已取得较好效果。至于设计新的语言，由于涉及与现有软件的接口问题，在并行处理结构上继续利用人类积累的巨大大软件财富，仍然是一个悬而未决的难题。

并行机性能的有效发挥，在一定程度上依赖于操作系统有效地支持。目前并行机上的操作系统大多数是 Linux。并行操作系统与串行操作系统的主要差别在于前者在资源调度、进程同步和通信等方面比后者具有较强的功能。

3. 任务分割问题

任务分割问题一般是指如何将一个大任务分割成可执行的多个子任务，把程序中固有的并行性尽可能地提取出来，以获得最快的执行速度。任务分割通常由用户、编译程序实现或在运行时由机器实现。用户在编程时用并行控制语句提取并行性是目前较流行的方法，也是最原始的方法。针对循环的并行处理是目前用得最多的，一般将内层循环向量化，外层循环多重处理。

任务分割影响并行处理的粒度，而粒度与负载平衡和开销又是相关的。任务的运行性能和子任务计算时间与子任务间通信时间的比率（即粒度）有很大关系。比率越高，子任务粒度越粗，通信与同步开销越小，但并行速度也越低，负载平衡也较困难；比率越低，并行度提高了（即粒度细了），但开销随之增大。因此，两个极端都不能获得高处理速度。较理想的方案是编程或编译时进行分割，运行时又允许子任务组合成大任务，以实现并行度和开销之间的平衡。

4. 任务调度问题

如何把一个程序的多个子任务最优地分配到并行处理的计算机中，从而尽可能地缩短并

行执行时间是调度的主要内容,也是有效地使用大规模并行处理计算机系统的一个关键问题。可以将一个程序的相互有影响、可并行执行的多个子任务分配给一个或多个处理机,以获得该程序的高性能或多道程序的高利用率。

调度有两种:静态的或动态的。静态调度是根据全局信息,在编程或编译时把子任务分配给指定的处理机,并在运行时不变化。静态调度的优点是运行时没有调度开销,特别适用于只需调度一次、程序不变而多次对不同数据求解的应用题目。其缺点是程序运行前无法确切知道各进程的运行时间,从而导致系统负载不易平衡,处理机利用率不高。动态调度是在运行时由机器根据各个处理机负载的轻重而给它们分配进程。动态调度需要由专用硬件或操作系统功能的支持。硬件支持的缺点是成本高,通用性差;操作系统支持的缺点是调度开销大。通常动态与静态相结合的调度效果较好。调度是一个很复杂的问题,由于目前的任务分割方案大多是针对循环的并行执行,因此很多调度算法也是针对循环并行执行的。

5. 同步问题

同步是保证相互协作的多个进程按正确的顺序执行,以获得串行执行结果的方法。松散耦合的多处理机系统多数采用消息传递方式,而紧密耦合的多处理机系统多数采用共享变量方式。

共享变量是通过某种形式的指令对共享变量以不可分割的方式执行读、改、写操作来实现的。具体方式可采用“忙-等待”策略、主动唤醒策略或它们的组合。在“忙-等待”方式中,进程对同步变量在未达到其所要求的状态之前进行循环检测。其优点是实现简单,比消息传递方式迅速,可用于细粒度进程之间的同步。其缺点是增加互连网络和共享存储器的竞争。在主动唤醒方式中,进程在检测到同步变量未达到它所需求的状态后便自动挂起,进入睡眠状态或切换进程,直到别的进程释放该变量并唤醒它为止。该方式要求系统必须提供一种机制来检测同步变量的状态变化,挑选和唤醒处于睡眠状态的进程。在组合方式中,进程阻塞自己以等待状态,当进程被唤醒后,它重新检测共享变量的状态,以决定向前执行还是重新阻塞自己。

任务的分割、调度和同步三者之间是相互联系、相互影响的。分割的粒度越细,并行性越高,则调度和同步开销也越大,程序执行时间不一定会减少;分割的粒度越粗,调度和同步开销减少,但并行性降低,程序执行速度不一定会提高。因此,最佳并行性(即最佳处理机数目与并行执行时间)和调度及同步开销之间有密切关系,二者应达到平衡才能取得最佳效果。

10.3.4 并行算法的效率与并行机系统结构的关系

并行算法的效率与并行计算机的系统结构密切相关,在算法构造时必须考虑它们之间的关系。

(1) 并行化的粒度。指算法中可并发运行部分的大小和复杂性。在粗粒度并行算法中,算法分成可并发执行的大程序块。它适合于多处理机系统,也适合于阵列机,当处理机之间的通信区域较小时,甚至也适合于计算机局部网络。在细粒度并行算法中,单独的指令或小的指令组即可并行执行。该算法最适合于向量机,因为计算大矩阵和长向量时,可以串行地计算梯度分量,并且将其中的计算向量化,形成细粒度并行算法。

(2) 通信要求。指并发过程之间需要共享的数据量,或者指某些过程的运行对其他过程的依赖性。对多处理机系统的并行算法通常设置整体变量或利用消息传递进行通信。具有共

享存储器和高速通信网络的多处理机系统适合于通信要求较高的算法，而计算机网络只适合于通信量较少的计算。

（3）运算的一致性。指并行执行的运算是否相同。向量机要求算法运算具有高度的一致性，即要求同样的运算重复地分配到向量的各个分量上。阵列机也要求具有较高的一致性，同样的指令序列可前后紧凑地对多重数据进行运算。多处理机系统和计算机网络可以并发地执行不同的指令，对一致性要求较低。

（4）过程的生成。指确定在何时何处生成过程。并行算法可以静态或动态地生成过程。静态是指在程序运行之前就知道过程如何生成；动态是指在算法运行过程中随着运行结果的变化而确定如何生成过程。静态生成过程可减少系统开销；动态生成过程可提高算法性能，但由于程序模块的加载等因素会使系统开销急剧增加。

（5）过程的控制。指过程运行对于其他过程进展的依赖性。若并行算法中某些过程段的开始依赖于其他过程的过程段的结束，称它为同步算法，否则称为异步算法。阵列机上运行的算法是高度同步的，而在多处理机系统和计算机网络上运行的算法既可以是同步的也可以是异步的。

由上述内容可以看出，并行算法研究与串行算法研究的一个重要的不同点在于，它是以新的方式与计算机系统结构相互作用。对串行算法而言，算法研究与硬件是相对独立的，算法优劣独立于串行计算机的结构。但对于并行算法来说，它强烈地依赖于并行计算机的系统结构。反过来，并行算法的研究也对计算机性能提出了新的要求，促使设计各种专用计算机来实现并行算法。算法与计算机在更广阔的范围和更深刻的程度上相互作用着，推动了计算机技术的发展。

10.4 面向对象程序设计的影响

当前，绝大多数的计算机属于冯·诺依曼系统结构，而冯·诺依曼系统结构的最根本特点是存储程序，以及由此而引申的顺序执行、集中控制、共享存储单元、指令和数据均为二进制表示并可修改等一系列特点。计算机的系统结构与所用的程序设计语言类型有密切联系。在冯·诺依曼结构的计算机上运行的程序设计语言，绝大多数是面向过程的命令式语言（Imperative Language）。

命令式语言的特点如下：

（1）用变量模拟计算机的存储单元，用赋值语句模拟对存储单元的存/取和算术、逻辑运算，用控制语句实现比较、测试、转移等操作。

（2）每执行一条赋值语句只产生对应于一个存储单元的结果值，命令式语言从本质上来说是顺序型的，赋值语句成了这类程序设计语言的瓶颈。

（3）由于在程序计数器（Program Counter）集中控制下逐条指令顺序执行，所以限制了操作并行性和分散控制潜在优势的发挥。

（4）当多个程序共享同一存储单元时，程序并行执行时的同步处理复杂化，从而难以设计出高度分散的计算机系统。

虽然出现了流水线处理机、并行处理机、相联处理机、多处理机和分布式处理机等，它们发展和改进了冯·诺依曼结构，不同程度地开发了计算机的并行度，但本质上仍然维持存储程序型的顺序操作概念，很难最大限度地挖掘计算的并行性。因此，除了对命令式语言继

续改进外，还提出了若干非冯·诺依曼结构的程序设计语言，并建立了适合于这类语言的计算机系统结构。面向对象程序设计语言的出现，对计算机系统结构提出了新的要求，促使其有了新的发展。

10.4.1 面向对象的程序设计

一个程序主要由数据和实施算法的过程两部分组成，二者有着直接联系。执行一个程序就是激活一个过程去对某些数据进行操作。当程序很大和很复杂时，各种类型的数据及数据结构与起不同作用的过程之间的联系变得非常错综复杂，从而使程序设计难度增大，使设计效率降低，也使软件的可靠性、可读性和可维护性显著变差。为了弥补面向过程程序设计的不足，人们发展了模块化结构的程序设计思想和方法：把一个庞大而复杂的程序系统分成相对独立的若干模块，每个模块定义相应的输入、输出界面，由多个程序员分工合作编写，齐头并进，提高编程效率；同时，每个模块只引入少数几种控制结构，简化程序模块的设计。模块化结构程序设计虽然提高了软件的可靠性和开发效率，增强了程序的可读性和可维护性，但是不能显著地改善程序系统整体上的复杂性。因为数据结构和过程仍是两个相对独立的实体，相互之间又有许多直接的联系，而且一种数据或数据结构经常为多个过程共享，一旦数据或数据结构发生变动，会严重影响使用该数据、数据结构的过程或程序模块的编写。因此，编程人员不得不重点关注数据组织和数据结构。

图 10-9 所示给出了模块化结构程序设计与面向对象程序设计的区别。图 10-9 (a) 所示模块化程序设计中的数据结构 1, 2, 3, 4 均独立于各个分程序模块，每个分程序模块都可以直接对各个数据结构进行操作。分程序模块和数据结构两类实体之间有着错综复杂的相互联系。整个程序由总控程序模块控制其操作顺序，若某个数据结构需要修改或扩展时，就需要对所有与其有关联的分程序模块进行修改，甚至重写。由于各分程序模块都要对与己有关的数据结构进行操作，从而使程序模块间的输入、输出界面不够清晰，在系统联调时往往不易处理程序模块间、程序模块与数据结构间出现的问题。

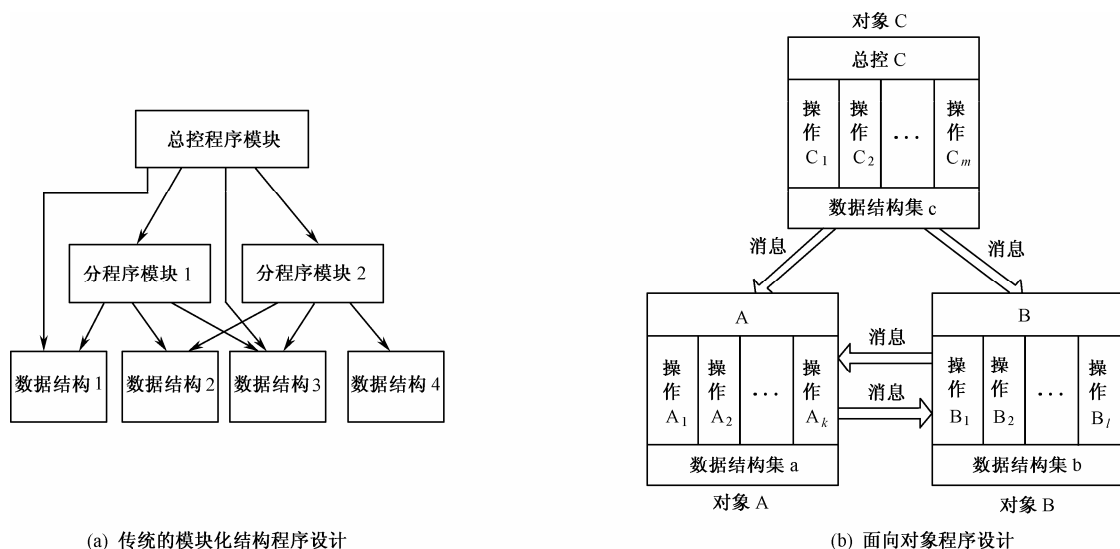


图 10-9 模块化结构程序设计与面向对象程序设计的区别

可以把数据和过程这两个逻辑上独立的实体组合在一个逻辑实体即对象（Object）中，使对象之间只通过发送消息进行联系，如图 10-9（b）所示。对象 A 给对象 B 发送一个包含要求对象 B 对其某一个数据或数据结构进行某种处理的消息，对象 B 予以响应，进行相应处理后发回结果给对象 A，而对象 B 内部具体操作对于对象 A 是“透明”的，对象 A 的某个过程也不能直接对对象 B 中的数据进行操作和处理。反之，对象 B 对对象 A 也是如此。这样，从整体上就可以降低程序系统的复杂性，增强可扩展性和修改的灵活性，有利于提高软件开发效率。图 10-9（b）所示的程序系统由总控 C，A，B 三个对象组成。每个对象内部都有自己定义的操作、过程、函数和数据结构。对数据结构或相应程序段的修改、扩展，只局限于所在对象内部，不影响其他对象，软件产品的灵活性大大加强。对象之间的界面简单清晰，只存在传输消息的通信联系，各对象之间具有很强的独立性，因而便于设计、调试、检错和维护，有利于各对象的同时编写，减少相互协调对接，提高了软件开发效率和管理水平，更符合软件工程的要求。

由对象组成的程序系统既可以在单处理机上运行，也可以以一个或一组对象为相对独立的工作单位，同时分配给多个处理机执行，充分利用多处理机并行处理的优点。

20 世纪 70 年代出现了软件集成化概念。软件集成化指系统程序员提供封装紧密的功能模块，它与具体应用无关，但可相互组合，实现具体应用功能，并能重复使用。面向对象程序设计是其发展中的重要一步，经过多年开发，产生了多种面向对象的程序设计语言，其中 Smalltalk-80 是这类语言的典型代表。Alan C. Kay 于 1972 年 9 月发表了“面向对象”的核心，几经修改、发展，演化成现在的形式，Smalltalk-80 是影响最大的版本。

在 Smalltalk-80 语言中，每个对象都被看作一个实例（Instance），具有某些相同属性的实例又归成类，用类属（Class）描述。每一个实例有一组实例变量（Instance Variable）。每一类属中有一组与这些实例变量相对应的实例变量名（Names）和一组方法（Methods）。每一个方法用消息模式、临时变量名和表达式组成，用以描述一个动作序列。当属于某一类属的实例收到一个消息时，将按消息的要求激活一个该接收者所属类属的方法，以完成一系列动作。“消息”是通过表达式描述的，表达式是关于该消息的接收者、选择符和参量的描述。消息的基本结构有下列 4 种：① 单元消息（接收者 + 选择符）；② 二元消息（接收者 + 一元选择符 + 参量）；③ 单参变量关键字消息（接收者 + 关键字 + 参变量）；④ 多参变量关键字消息（接收者 + 关键字组 + 参变量组）。类属之间存在继承性。一个类属或实例可继承另一个类属或实例的主要属性，又可具有自己的属性，增加新的实例变量、方法和类属变量，成为超类属（Superclass）的子类属（Subclass）。Smalltalk-80 是开创面向对象程序设计范例的先驱，它把整个编程环境和基于选单的交互式界面集成起来，对整个软件的发展产生了深远的影响。Apple 公司的 Macintosh，Microsoft 公司的 Windows 都受其启发和引导，其中都能看到它的“影子”。

10.4.2 基于面向对象程序设计语言的计算机系统结构

由于面向对象程序设计语言的程序中不需要说明变量类型，子程序（过程）在程序运行过程中自行链接，因此具有很高的编译效率和自动管理动态变化的数据结构的能力。所以面向对象程序设计具有较高的软件生产效率。但是，要充分发挥这种程序语言的效能，在传统的计算机上是难以全面体现的，这是因为传统计算机：

- ① 没有数据类型说明，在运算操作前必须对操作数进行校验。
- ② 程序中小过程的数目大大增加，调用次数多。
- ③ 过程调用中必须通过查表解决参数类型是否一致，以及增加过程调用的开销。
- ④ 要对动态变化的数据结构实现自动管理，就会增加运行中存储空间的再定位和回收等操作的次数。

所以，需要设计新的计算机系统结构，以适应面向对象程序设计语言，充分发挥其效能。

1. 面向对象系统的基本特点

(1) 用框架 (Frame) 知识表示法中的框架概念表达某个对象，每个对象 (框架) 都有特定的标识符命令，有关操作以及相应的数据结构或知识结构隐含在对象中。

(2) 在消息传递通信中，对象之间是通过标识符或操作符的模式匹配进行控制转移的。各对象之间通过发送或接收信息相互联系。

(3) 因为对象是把数据结构和对数据进行操作的过程合为一体的逻辑实体，因此在计算机上实现时，对象实际上要占据一段存储空间，具有统一格式的数据结构。

2. 面向对象程序设计语言的计算机结构

(1) 具有虚拟存储系统，能提供完善、高效的面向对象的动态存储管理、存储保护及快速匹配检索对象的功能及其硬件支持。

(2) 它是一个多处理机系统，使各个对象或若干对象组合的模块能在各自分配到的处理机上执行，提高并行处理能力。

(3) 具有对象之间消息传递的通信系统。

3. 举例

美国 Intel 公司 1981 年推向市场的 iAPX432 系统是第一个基于面向对象程序设计的 32 位多处理器系统，采用 CISC 技术，其结构框图如图 10-10 所示。

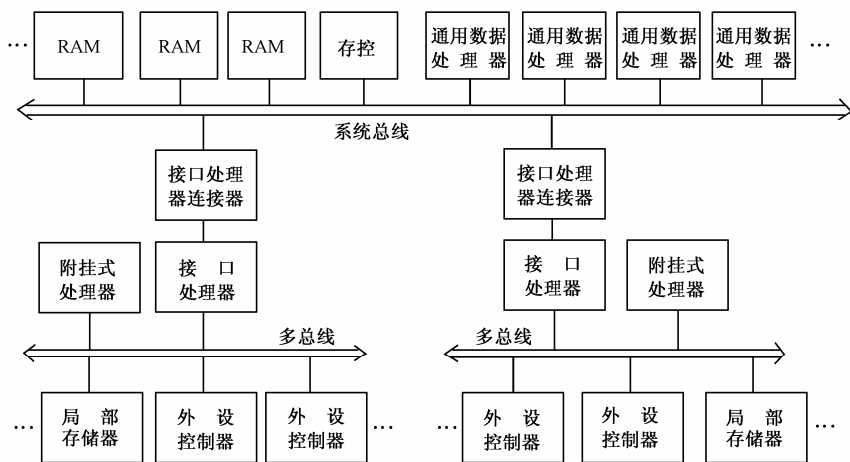


图 10-10 iAPX432 系统结构

通用数据处理器 (GDP)、主存模块 (RAM)、接口处理器 (IP) 经系统总线互连，它们的个数可以根据用户需求增减。GDP 是 VLSI 芯片，内有 80 位浮点运算部件可加快浮点运算。IP 通过多总线与局部存储器、外设控制器、外设构成 I/O 子系统。该系统采用 Intel 公司的

MULTIBUS 标准。GDP 和 IP 有主机/校验两种工作方式，可并接成“双工”系统以自动发现硬件故障，使该系统具有较高的可靠性。

iAPX432 系统的机器指令设计接近于 Ada 语言的语句，许多机器指令与 Ada 执行语句是对应的，采用面向编译的硬件结构。特别是利用 Ada 语言提供的并发程序设计功能大大简化了并行程序的设计，支持多个 GDP 并行工作。

iAPX432 系统的对象相当于 Ada 语言中的一个“包”（Package），用标识符标记，内部由



图 10-11 iAPX432 对象结构

包说明和包体两部分组成，如图 10-11 所示。包说明有过程名/函数名及参数定义，用于同其他用户或包的通信接口。包体由数据结构集合及功能和操作构成。iAPX432 程序的基本结构包含处理器、进程、上下文、指令、数据 5 类对象。一个对象可占据 1~64KB 范围内动态变化的线性逻辑存储空间，用对象起始地址加位移量（字节数）确定元素的物理地址。

iAPX432 系统的一条指令由操作码、位移量及地址字段组成。访问某个对象时的具体步骤如下：

- （1）按 GDP 内的地址 Cache 中地址读出“访问描述符”，字长 32 位，内含有关对象的标识和访问权限等信息。
- （2）在满足访问权限的前提下，根据对象标识再到系统地址空间找出相应的“对象描述符”，由两个 32 位的字组成，内含对象类型、基地址、长度、虚存控制等信息。
- （3）当自动校验类型相符且指令中的位移量不超出对象长度范围时，就把指令中位移量与对象基地址相加得到物理地址，访问相应存储单元。

由此可见，寻址硬件本身有较强的存储保护功能，将面向对象的设计思想与硬件寻址部件结合起来，可使信息保护到数据结构级，使程序保护到过程或函数级。

iAPX432 系统提供硬件支持的通信口对象和发送（Send）、接收（Receive）两条指令，解决并发进程间的通信和同步。通信口对象起两个异步进程间缓冲器的作用，允许任何一种对象作为信息经通信口对象传送。

iAPX432 操作系统 iMAX 用 Ada 语言编写，操作系统的进程调度、存储分配、进程同步和通信已硬化，从而使操作系统的设计大大简化，运行速度加快。

iAPX432 系统属于紧耦合型，也可采用松耦合型结构。例如，局域网（LAN）、机群系统等，以客户-服务器（Client/Server）结构进行分布式计算，实现面向对象的程序设计语言。

10.5 软件的固化与硬化

计算机系统是由硬件和软件组成的。软件随着计算机系统的应用领域不断扩大和深入而迅速地丰富起来。例如，操作系统的日益发展，为方便使用者提供了众多的功能，只需发出少数几条指令，就能使计算机系统执行复杂的动作。但是，随着半导体器件制造工艺的不断发展，集成度、运算速度不断提高，价格大幅度下降，单个芯片的功能也有惊人的进展，微处理器的发展历程就是最明显的例证。所以计算机系统结构设计必须考虑的软、硬件分界面

在不断变化着。例如，早期机器乘/除法运算是通过子程序（软件）实现的，而目前机器绝大部分均已将乘/除法软件“硬化”，即用硬件实现乘/除运算。又如，一般机器只有整数（定点数）、实数（浮点数）和逻辑数的数据表示，对于向量、数组等复杂数据结构只能用软件实现，而现在有些机器已具有向量、数组的硬件表示以及相应的运算功能。其他如中断处理、存储管理等软、硬功能分配也随着不同时期、不同机器而动态地改变着。其基本趋势是软件向硬件转化。例如，目前 32 位的微处理器多数具有完善的中断处理功能和存储管理部件。

在硬件设计中，应尽可能选用大量生产的高集成度的通用芯片，有效途径之一是如何更多地采用存储逻辑。采用存储逻辑是指用 VLSI 技术（即超大规模集成电路技术）生产的存储芯片，配以实现特定功能的、以机器语言编写的程序，替代原来由组合逻辑所完成的功能。微程序就是其中的一个典型例子。微程序技术能把组合逻辑控制网络的复杂逻辑结构转变为控制存储器的各种可执行的代码。一般采用 PROM 来存放微程序，而 PROM 芯片通用性很高，可大量生产。用查表法实现数制、码制变换，或者参数的运算，是另一个例子。随着存储芯片存、取速度提高和价格不断下降，为查表法的实现提供了可能。又如，为了提高操作系统的运行速度，将基本输入、输出系统（即 BIOS）存放于 EPROM 内，以固件的形态出现于计算机系统中。各类例子都有一个共同点：将具有某种功能的可执行的程序代码存放于掉电时也不丢失信息的存储芯片（如前述各种 ROM 型芯片）内，以固态方式呈现，这就是软件固化。软件固化在系统硬件设计中已普遍使用。可编程逻辑阵列（PLA）芯片也是存储逻辑中的一种。它灵活性大，能以比存储芯片少的“冗余”量实现逻辑要求，延迟时间仅在数十纳秒内，因此，目前有众多的机器用 PLA 实现系统控制部件。软件固化的进一步发展，将减少计算机系统结构的层次，将出现由微程序直接实现高级语言的所谓高级语言计算机，由固件直接实现操作系统的所谓操作系统计算机。

由上述内容可见，软件和硬件在逻辑功能上是等效的。由软件实现的操作（如编译、解释、操作系统的基本命令等）在原理上是可以硬化成硬件来实现的；同样，由硬件实现的操作（如某条机器指令的功能）在原理上也是可以用软件的模拟来实现的。由自动机的基本理论可知，只要机器有“相减”及“转移”这两种指令就可用于算题、求解。因此，具有相同功能的计算机系统其软、硬件功能分配可以在很宽的范围内变化，如图 10-12 所示。选择怎样的分配比例，主要取决于现有硬件状况（主要指逻辑和存储的芯片状态）下的性能价格比。提高硬件功能的比例可以提高运算速度，减小所需存储容量，但会提高硬件成本以及降低硬件的利用率和计算机系统的灵活性与适应性；相反，提高软件功能的比例可以降低硬件的造价，提高灵活性和适应性，但运行速度要下降，所需存储容量要增加。所以，软、硬件功能分配是在现有硬件状况、运算方法和解题算法基础上的动态平衡。

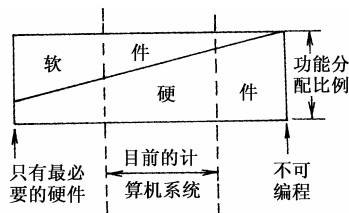


图 10-12 计算机系统的软、硬件功能分配

第 11 章 现代计算机系统结构的发展

随着科学技术的飞速发展以及计算机应用领域的日益扩大，对计算机系统的处理能力和计算速度提出了更高、更新的要求。为了大幅度提高计算机并行处理的能力，在计算机系统结构技术上必须有所突破，寻求有利于开发高度并行功能的计算模型和相应的现代计算机系统结构。

11.1 集群计算机系统结构

11.1.1 集群计算机系统及其特点

无论是 PVP，MPP，NCC-NUMA，CC-NUMA 还是 SMP，其共性是：需要从底层到顶层实现整个计算机。它们的开发是芯片级的，技术难度大，硬件研制周期很长；另外，由于缺乏通用的商业软件的支持，需要重写系统软件，软件研制周期也很长。这些因素除了增加研制成本外，还会降低高性能计算机的竞争力，因为在研制开始时所使用的先进的芯片技术，在研制结束时却会因为周期太长而失去了领先性。

相比之下，集群（Cluster）计算机系统能够以较短的研制周期、集成最新技术、汇集多台计算机的力量，达到较高的性能价格比，其技术发展在国际上受到重视。它通过高速互连网络把通用计算机（如高档计算机、工作站或 PC）连接起来，采用消息传递机制（MPI，PVM 等），向最终用户提供单一并行编程环境和计算资源，因此它通常也称为“计算机群”、“工作站群”、“工作站网络”或“网络并行计算”等。

集群计算机技术是在 20 世纪 90 年代中期才开始发展的，但其发展速度非常迅猛，我们从国外高性能计算机权威网站公布的“全球 500 强（TOP500）”排行榜可以清楚地看到它的发展趋势。1997 年 6 月时只有 1 台 Cluster 进入了 TOP500 排行榜，而到 2000 年 11 月已有 28 台。无论其数量、所使用的处理器数目、峰值速度，还是 LINPACK 指标，都呈现指数增长，如表 11-1 所示。

表 11-1 TOP500 中 Cluster 的数量

比较项目 \ 时间	1997年 6 月	1997年 11月	1998年 6 月	1998年 11月	1999年 6 月	1999年 11月	2000年 6 月	2000年 11月
数量（台）	1	1	1	2	6	7	11	28
所使用的处理器数目（个）	100	100	68	290	1026	2412	3668	9174
峰值速度（GFLOPS）	33	33	72	299	788	2007	3720	8717
LINPACK 指标（GFLOPS）	10	10	19	102	370	959	2254	5409.8

最初，集群计算机系统的使用效率只有 30%左右，严重影响它的发展。随着高速通信、体系结构、并行算法等方面研究的不断深入，集群计算机系统的使用效率取得了比较满意的结果，已达 62%以上，逐步接近 MPP 的效率，如表 11-2 所示。

表 11-2 高性能计算机的效率比较

比较项目 \ 时间	1997年 6月	1997年 11月	1998年 6月	1998年 11月	1999年 6月	1999年 11月	2000年 6月	2000年 11月
Cluster	30.30%	30.30%	26.39%	34.11%	46.95%	47.78%	60.59%	62.06%
MPP	67.46%	67.09%	68.79%	67.89%	67.33%	66.04%	68.20%	76.14%
Constellations	49.15%	64.13%	72.92%	43.74%	63.17%	57.38%	58.52%	69.15%
SMP	79.09%	82.69%	81.60%	80.73%	76.73%	83.69%	84.89%	90.07%
TOP500	68.84%	69.24%	70.59%	66.17%	67.60%	65.86%	67.86%	74.50%

集群计算机可以从狭义和广义的角度进行划分。狭义地讲，根据构成结点的不同，可以分为 PC Cluster 和 工作站 Cluster。根据研制理念的不同，可以分为 NOW 类型 Cluster（追求高速通信，进行全局资源管理，采用时钟周期“窃取”技术来利用空闲计算机的资源）和 Beowulf 类型 Cluster（尽可能使用现成的硬件、免费系统软件、基于 TCP/IP 建立通信库、不考虑“窃取”时钟周期）。广义地讲，由 Cluster 本身作为结点构成的 Cluster（称为 Hypercluster）、由 SMP 结点构成的 Cluster（称为 CLUMPS 或 Constellations）、ASCI 机以及元计算网络，都可称为 Cluster。美国最有影响的集群计算机系统是 Berkeley 大学 1997 年推出的 NOW 系统。它由 100 个 SUN 工作站、用速度为 160MB/s 的 Myrinet 连接而成，峰值速度为每秒 330 亿次浮点运算。集群计算机系统的应用面非常广，除了科学计算外，还可以用于事务处理，如用作 Web 服务器、网络文件服务器、超级 Mail 服务器以及海量廉价存储系统等。集群计算机的基本结构如图 11-1 所示。

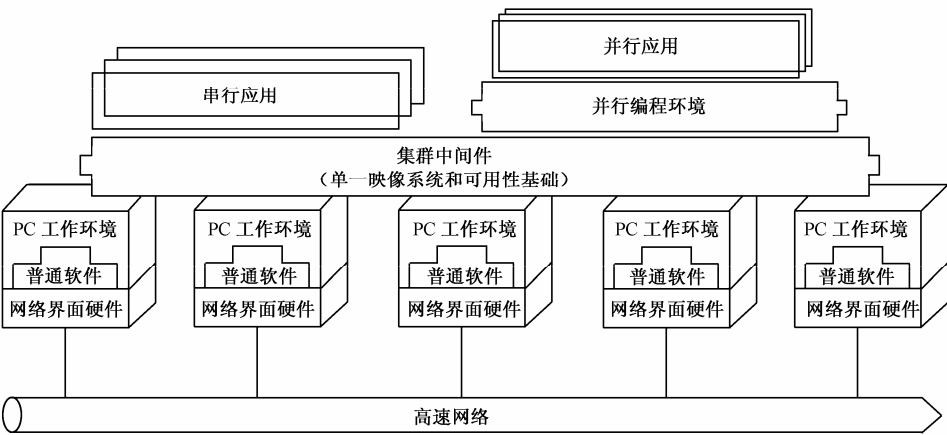


图 11-1 集群计算机的基本结构

我国在研制高性能计算机方面，已经取得很多成就。这些高性能计算机主要划分为如下三大类：

- （1）PVP 向量型超级计算机，如国防科技大学研制的银河 I（1 亿次/秒）、银河 II（10 亿次/秒）。
- （2）MPP 大规模并行处理超级计算机，如国防科技大学研制的银河 III（130 亿次/秒）、

中国科学院计算技术研究所研制的曙光 1000（25 亿次/秒）、中国江南计算技术研究所研制的神威 I（3840 亿次/秒）。

（3）集群计算机，中国科学院计算技术研究所研制的曙光 2000-II（1100 亿次/秒）、曙光 3000（4030 亿次/秒）、清华大学研制的 THNPSC-1（320 亿次/秒）、上海大学研制的自强 2000（4500 亿次/秒）。

可以看出，国内高性能计算机研制的重点集中在 MPP 和 Cluster 上。MPP 主要集中在军口，Cluster 主要集中在民口。

11.1.2 集群系统的通信软件和网络服务

集群系统的高效通信子系统是十分重要的，因为集群系统有更高的结点复杂性，结点之间物理线路很长，导致互连网络延迟很长。同时，也带来了线路可靠性、安全性问题，它们都需要由通信协议及其实现的通信软件和网络服务来解决。分布式应用程序所需的通信是多样化的，从点到点通信到多播通信，且能支持成批数据传输、流数据、组数据等。在一般操作系统中，套接字方式可以在消息传递方式里用来进行进程间通信，但是在集群计算机中，它的通信软件使用专用网络的集群计算机通信协议，提供快速而可靠的结点间以及与外界数据通信的手段。它往往使用绕开操作系统的轻量级通信协议，来消除操作系统对系统性能影响很大的额外通信开销。集群计算机的通信服务调用提供了集群计算机传输管理和用户数据所需的基本机制，并且保证延迟、带宽、可靠性、容错等服务质量，提供对用户接口的直接用户层访问。如通常集群计算机采用相对低层的应用程序接口 API（Application Program Interface）来支持更大范围的高层通信函数库和协议。

11.1.3 集群系统的资源管理和调度

资源管理和调度是使集群计算机系统的分布应用程序达到最大吞吐量的操作，使资源的有效和高效利用成为可能。一般而言，它由三部分组成：

- （1）用户服务器。它允许用户向一个或多个队列提交作业，指出每个作业的资源需求，从队列中删除一个作业、询问一个作业或一个队列的状态信息。
- （2）作业调度器。它根据作业类型、资源需求、资源可用性以及调度策略来执行作业调度和排队。
- （3）资源管理器。它监视和分配资源、执行调度策略、收集记账信息、实现作业迁移和检查点操作。

11.1.4 集群系统的单一系统映像

一组由以太网（Ethernet）连接起来的工作站不一定就是一个集群系统，一个集群系统是一个单一系统。单一系统映像 SSI（Single System Image）是为了让用户所看到的一个集群系统在使用、控制和维护上就像一个 PC 机一样。单一系统映像是一个内容十分丰富的概念，包括单一入口点、单一文件层次结构、单一 I/O 空间、单一网络、单一作业管理系统、单一存储空间和单一进程空间等。例如，单一入口点是使用户能像登录一个虚拟主机那样登录集群系统，尽管集群系统中可能有多个物理主机结点为登录服务。集群系统可以透明地将用户的登录和连接请求分布到不同的物理主机上去，以达到负载均衡。

对单一系统映像而言，在逻辑上是单一控制；在位置上是透明的，用户不用知道提供某个服务的物理设备的具体位置；在使用上是平等的，对所有结点和所有用户都可以使用集群系统中的服务和功能。

11.1.5 集群系统的并程序序设计环境

为了充分利用集群系统的资源，用户要解决的问题的算法描述必须由一组可以并发执行的子问题和子任务组成。而它们必须通过使用并行模型、并行语言以及并行环境来实现。集群系统上的并程序序设计环境大致有两种模型：

(1) 整个存储器空间对于集群系统中任何一个结点都是全局可寻址的，称为共享存储器模型。特别是分布式共享存储器 DSM (Distributed Share Memory) 模型是低层的硬件和在其上面运行的操作系统一起负责把分布式存储空间转换成全局可寻址的单一存储空间。常用的技术是信号量、条件临界区和管程。

(2) 在分布式存储器中进行任务间通信的消息传递模型是目前使用最广泛和最有效的并程序序设计环境。常用的技术是套接字、远程过程调用和汇合机制。比较著名的有 PVM 和 MPI，它们都有消息传递库。

11.2 高性能计算机系统实例

上海大学从 1999 年起开始研制集群式高性能计算机系统，于 2000 年 9 月完成第一期工作，完成自强 2000 集群式高性能计算机系统；2001 年 4 月完成第二期系统扩展和提速，使整个系统的 CPU 数达到 221 个，速度峰值达到 4500 亿次浮点运算/秒 (450GFLOPS)。

11.2.1 自强 2000 的体系结构

自强 2000 系统划分为三大部分：主体系统、辅机系统和前置处理系统，如图 11-2 所示。其中，主体系统有一个高速紧凑核心，它由 1.28Gb/s 的 Myrinet 网络将 16 个高性能 SMP 结点连接而成，这些结点的内存大至为 512MB。可以看出，这个核心无论是从通信性能上，还是处理能力上，都具有很强的能力，它对于解决一些不需要很大规模但有较高通信要求的问题具有良好的性能。主体系统的其他 52 个结点主要是由 800MHz 和 700MHz Pentium III CPU 构成的 SMP 结点，连接方式为 100MHz 的快速以太网。主体系统的 CPU 总数达到 136 个，具有较大的规模和较均衡的处理能力，适合处理一些通信要求适中的大规模问题。

前置处理系统由 8 个结点，共 16 个 CPU 构成。它们既可以用作 Web 服务器，处理来自 Internet 的访问请求，又可以用来参与高性能计算。

辅机系统由 30 个结点、60 个 CPU 构成，主要由 Pentium III 550 和 Pentium III 500 构成。辅机系统、前置处理系统和主体系统一起，用于计算的结点总数达到 106 个，CPU 数达到 212 个，适合计算一些特大型计算密集型问题。此外，系统配备了三个备用结点和三个用于防火墙等功能的辅助结点。整个系统的内存总容量超过 20000MB，硬盘总容量超过 2000GB。

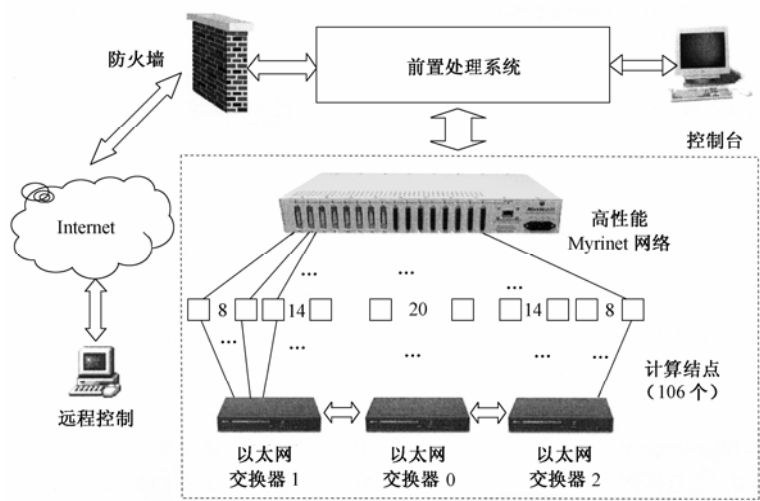


图 11-2 自强 2000 的体系结构

11.2.2 自强 2000 的软件环境及其特点

(1) 面向世界网络时代，为用户提供基于 TCP/IP 的 Web 结构来管理和访问。它提供 Web Browser 用户界面，使用户可以在远距离，通过 Internet 来登录、查询计算结果和曲线，进行一定权限内的远距离管理。这种框架符合世界技术发展，并在以后还会有很多可发展的余地。它用 Java Servlet，通过 JDBC Driver 和数据库交互。用户通过 HTTP 协议获取和提交数据，查询计算结果，看到计算曲线。还可以让用户看到自己计算时的处理机利用率、内存利用率和通信开销等硬件系统资源利用情况。

(2) 一般高性能计算机都采用文件来存储中间计算数据和结果数据，不便于查询，查询的响应时间也无法保证。自强 2000 采用 SyBase11.9 关系数据库，不仅便于存储大量数据（包括中间数据），而且还可保证响应时间；它采用可交互、灵活指定绘图方式的 Java Client 绘制计算的中间结果曲线图形。使这些数据显示出规律性，并可通过数据挖掘来提炼高性能计算中有特性和有共性的数据。这对于推广高性能计算应用，构建高性能计算的“应用库”，并把高性能计算技术提高到更为成熟的阶段，极为有用。

(3) 高性能计算的推广应用，要十分注意并行算法和系统结构的匹配。而算法和系统之间的“桥梁”就是可视化的性能评价工具 PT (Performance Tool)。用自强 2000 解题时，处理机和存储器的利用率以及通信开销的实际情况用可视化形式告诉用户，使用户可以根据这些信息，修改其算法，优化计算结果。在用户并行算法和并程序执行可视化的情况下，使超级计算机系统、软件程序和并行算法这三者更密切结合，改变了过去用户与系统硬件资源距离很大的局面，使用户可以根据硬件系统的特点，调整自己的并行算法，以取得优化计算效果。

(4) HPF 是一种数据并行语言，统计表明，80%的科学和工程计算问题是数据并行问题。HPF 是国际标准语言 ISO/IEC 1539: 1991 标准 FORTRAN 90 程序设计语言的扩充，在 FORTRAN 90 的基础上增加了通用的数组并行运算语句 FORALL 及数据映射指导语句，使 FORTRAN 90 的数组运算功能得到进一步加强，表达更加灵活自然。数据分布指导语句使编

译者从系统相关的通信语句中解脱出来，使集群式系统以消息传递为主的低层编程模式提高了一步。

(5) 考虑到要把高性能计算推向更为成熟的阶段，其主要的特征是使应用经验“模块化”成“预构件”，即所谓的“应用库”。在自强 2000 中，十分注意积累高性能计算的经验和结果，并把这些构造成高性能计算的“应用库”，使用户可以用已有的“预构件”较快地完成高性能计算任务。

11.2.3 高性能计算应用举例

高性能计算能把科学研究和工程技术推上一个新的台阶。而且，高性能计算机是一个比较通用的平台，它可以托起理科、工科、军事、甚至社会学科领域的研究和设计水平。因此，高性能计算机的生命力在于应用。华东理工大学采用的计算机分子模拟技术是研究高分子系统相分离、微相结构及其随时间演化的重要手段。它根据分子结构和分子间相互作用直接模拟得到高分子系统的相分离和微相结构，可以排除实验中的许多不确定因素和实验系统选择的困难，同时又可以避免理论分析常常会碰到的数学分析上的困难。该研究课题在自强 2000 上开展了高分子系统的计算机分子模拟工作，对高分子膜的微相结构及其演化进行了模拟，取得了令人满意的计算效果。以往模拟高分子在平板狭缝中的分布结构时，高分子的链长最多能达到 64 个链节（国际上最好的是 201 个链节），需要几天，甚至几个星期的计算时间才能得到一个密度下的模拟结果。现在在自强 2000 上计算只需几个小时，并且高分子的链长也大大提高，其可视化结果如图 11-3 所示。

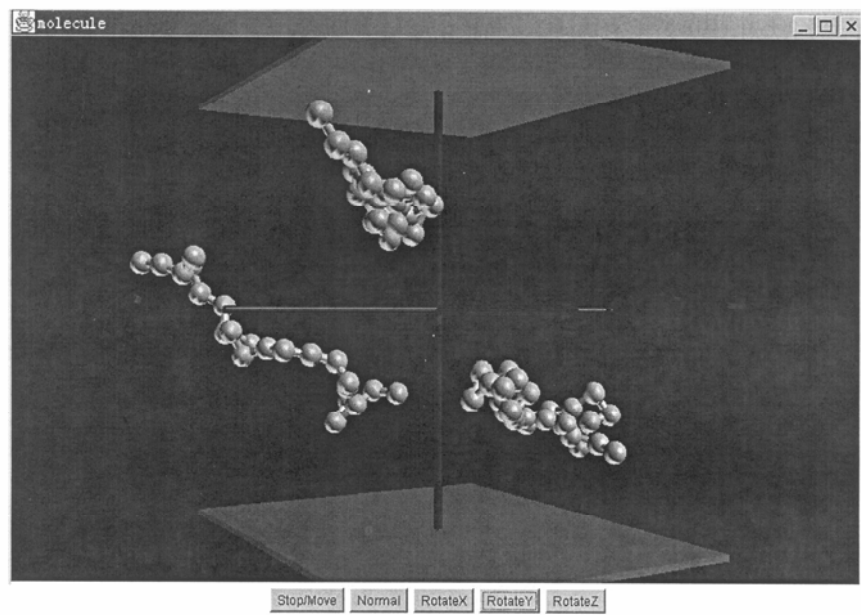


图 11-3 高分子材料微观结构的可视化结果

11.2.4 “天河一号”的诞生

进入 21 世纪，特别是近期高性能计算机发展从 T 级（百万亿次/秒）迅速进入 P 级（千

万亿次/秒），预计到 2015 年将进入 E 级（万万亿次/秒）。

2009 年 10 月 29 日，我国在长沙宣布了名为“天河一号”的第一台国产千万亿次高性能计算机的诞生，它以每秒 1206 万亿次（1.206PFLOPS）的峰值速度和每秒 563.1 万亿次（563.1TFLOPS）的 Linpack 实测性能成为 2009 年 11 月公布的全球高性能计算机 TOP500 排名榜（衡量系统性能）的第 3 位，也使中国成为继美国之后世界上第二个能够自主研制千万亿次高性能计算机的国家。同时，它十分注意环保节能的高效能，达到 431.7MFLOPS/W，排列于 Green500 排名榜（衡量性能功耗比，其单位是每瓦特百万次浮点运算）的前列。

“天河一号”采用 6114 个 Intel 至强（Xeon）多核服务器 CPU（2.53GHz 的 E5540 和 3GHz 的 E5450）和 5120 个 AMD 镭（Radeon）图形加速处理器 GPU（AMD-ATI4870 显卡），内存总容量 98TB，采用两级 Infiniband QDR 互连，单个通信链路的点对点通信带宽达 40Gb/s，而共享磁盘总容量达到 1PB。

“天河一号”采用了大量的具有自主知识产权的技术，如多阵列可配置协同并行体系结构、高速率扩展互连通信技术、高效异构协同计算技术、基于隔离的安全控制技术、虚拟化的网络计算支撑技术、多层次的大规模系统容错技术和系统能耗综合控制技术。

我国著名计算机专家周兴铭院士对“天河一号”的评价是，由于采用了一系列创新技术，这台计算机具有 4 大特点：

（1）高性能。无论是 1206 万亿次的峰值性能，还是 563.1 万亿次的 Linpack 实测性能，均位于国内榜首、世界一流。

（2）低能耗。能耗是每瓦电创造的计算效能，“天河一号”目前能效为每瓦 4.3 亿次运算，与 2009 年 6 月公布的 TOP500 排名第一的美国“走鹃”（Roadrunner）相当。参照 2009 年 6 月公布的 Green500 排名，“天河一号”可以位居第 5。

（3）高安全。“天河一号”实现了不同用户间数据和工作信息的相互隔离。对用户来说，相当于到银行租个保险柜，钥匙掌握在自己手里。

（4）易使用。“傻瓜化”的操作界面和菜单、鼠标等常规操作，让机器的使用变得简单。同时，作为一台国际通用的标准化超级计算机，在“天河一号”上能运行来自各种行业的各种程序，不存在兼容性问题。

“应用驱动”是高性能计算机发展的原动力。“天河一号”广泛应用于石油勘探数据处理、生物医药研究、航空航天装备研制、资源勘测和卫星遥感数据处理、金融工程数据分析、气象预报、气候预测、海洋环境数值模拟、短临地震预报、新材料开发和设计、土木工程设计、基础科学理论计算等方面。它将作为“国家超级计算天津中心”的业务主机，而向社会开放，实现资源共享，为国内外提供高性能计算服务。

11.3 网络技术

11.3.1 网络技术的基本概念

网格（Grid）技术，是 20 世纪 90 年代中期随着计算机网络技术和分布式计算技术的不断发展而诞生的一种全新技术。它以高性能网络为依托，借助于一套完善的网格中间件的支持，将分布于网络上的各种资源加以整合，为用户提供一套完善的具有单一映像的支持环境。在此基础上，网格使用者可以方便地对网格中的各种资源加以动态的有效利用，解决各

个不同领域中的科学、工程、商业等问题。在科学与工程领域，以网格中间件的支持为基础，研究者可以建立各种“虚拟组织”，打破行业、机构、地域的限制，开展大规模的合作研究。与传统的研究模式相比，基于网格系统进行合作研究具有组织结构灵活、信息传送迅速、研究协作高效、不受地理位置限制等优势，能够有效地促进研究合作，提高研究水平，方便研究成果共享，避免重复研究，提高研究资金效率等。除了面向科学与工程领域的应用之外，网格系统还在商业、信息服务等领域得到重视。网格技术已经成为下一代信息应用技术和处理技术的核心，被称为“下一代互联网”。

自从网格技术出现以来，网格相关的各种研究在全世界范围内得到了广泛的重视。美国、欧盟、日本等发达国家纷纷斥巨资支持对网格相关技术的研究，大规模的网格相关研究项目已经有上百个，其中比较有影响力的有 Globus Project, NPACI, NCSA, NASA IPG, Global Information Grid, e-Science, Data Grid, Tera Grid 等。网格技术已经成为对国家科技进步、国民经济发展、综合国力提高和国家安全具有重要意义的关键技术。在这种形势下，我国也已经认识到了网格的巨大作用。863 高科技计划已经启动了网格专项研究，力求在 5 年时间内，在网格结点建设、网格应用等方面跟踪国际先进水平并力争有所超越。同时，国家自然科学基金等国家、社会基金也开始对网格相关研究加以支持。国家教育部启动了“中国教育科技网格”(China Grid)，中科院计算所启动了“织女星网格”(Vega Grid)；上海市教委启动了“上海高校网格”(Shanghai Grid)等。

11.3.2 网格技术简介

网格技术模式是通过高速网络连接和统一各类不同物理位置的资源（超级计算机、大型数据库、存储设备、各种仪器设备、知识库等），配置系统软件、工具和应用环境，使之成为一个互相协调的先进计算设施。

它的主要研究内容如下。

1. 网格体系结构研究

网格体系结构的研究是研究网格技术和构建网格系统的关键。网格体系结构由三部分构成：网格分层、各层所提供的网格服务和为了提供这些网格服务所必须遵循的网格协议。如开放网格服务体系结构 OGSA (Open Grid Service Architecture)。

2. 网格资源访问规范

网格系统是建立在各种各样不同类型、不同平台、不同用途的资源基础之上的，这些资源需要以不同的手段、遵循不同的协议来访问。作为网格系统的基础，通用网格资源访问接口的作用就是为这些不同类型、不同平台、不同用途的网格资源提供一套统一的访问接口，通过此接口，各种资源对网格系统呈现一个统一的标准协议，从而为进一步整合网格系统中的资源提供基础。

3. 网格资源索引机制

网格资源索引系统为网格用户提供资源索引服务，是网格能够作为一个整体加以运行的关键，资源的分类与描述是资源索引的基础。网格资源分类法是以资源的语义特性为基础的，它需要维护一个标准的、可扩展的语义特性树，进而定义网格资源分类法则，完成网格资源的索引。

4. 网格数据管理规范

数据服务是网格系统的一个主要功能，在网格系统中存在大量的数据服务。为了把这些服务更好地提供给网格用户，网格系统需要提供可靠的、高效的数据管理机制，它包括数据复制、数据缓冲、数据一致性、数据存储管理等各种功能。通过提供这些功能，网格系统将在网格用户和网格资源之间提供一个可靠高效的数据通道。

5. 网格服务质量

网格系统的运行实质上就是各种服务被不断使用的过程。描述网格服务质量就是对一个资源的评价与计量，即资源的质量如何、资源的数量如何体现的问题。在通用网格中，由于资源类型的不确定性，建立一套完整的、适合于一切类型资源的通用网格资源评价与计量体系是很困难的。

6. 网络安全与网格用户管理机制

网络安全服务包括：数据存储和传输的安全、资源访问的安全、各种应用和相关数据的安全、用户信息的安全等。针对网格系统中对安全造成威胁的因素加以分析并研究相应的防范手段，最终为网格系统提供一套具有运行稳定、使用灵活、单点登录等优良特性的网络安全服务解决方案。网格用户管理技术要实现网格用户的认证与授权等功能。

7. 网格应用支持工具与开发环境

网格应用支持工具与开发环境为网格系统的用户提供一套能够比较简单有效地使用网格系统的各种资源来完成应用开发的工具与编程环境。它包括事务处理服务、故障处理服务、MPI 编程接口等。

8. 应用网格中的理论、模型、方法和算法研究

以网格的方式来解决应用系统的问题，必须要解决应用过程中所必须面对的各种理论、模型、方法和算法问题。必须研究在网格条件下用于解决资源优化和安全保证等问题的各种理论和模型，以及在此基础上研究新的方法和算法，如非线性优化算法、超大型线性规划算法、大型系统模型的分解算法等，这些方法和算法将能够更好地利用网格的特征，发挥网格的优势，解决传统方法不能或难以解决的问题。

11.3.3 网格技术应用举例

快速制造网格系统主要研究网格技术在制造业的应用。它是以上海大学、上海交通大学、同济大学、上海德尔福汽车空调有限公司、华中科技大学的各种快速制造资源，包括正向设计、逆向设计、快速原型制造（RP）、快速制模（RT）以及相应的技术服务（TS）等，作为不同的网格结点，以实现各种资源之间的互连互通、资源共享和协同工作，降低快速制造成本，提高快速制造资源的利用率为目标而建立的。

快速制造网格系统由以下 4 个层次构成：

（1）资源提供方（制造网格结点）。它利用现有的各种资源，包括高性能计算机、网络工作站、加工设备、测量仪器等。

（2）资源管理 Agent（设计 Agent，RP Agent，RT Agent，TS Agent）。包括资源请求 Agent，构件管理以及安全和可靠性管理三大模块。资源请求 Agent 有：

① 任务管理 Agent、任务调度 Agent 和任务分配 Agent。用于接收用户的任务和要求，调度各种资源，将任务分配到网格资源结点上运行，并随时把网格资源结点反馈的任务状态

或结果返回。

② 构件管理。包括网格资源搜索引擎、网格信息交换、资源管理与配置以及非技术因素管理（包括网格资源第三方认证机制和知识产权与利益分配机制）4 大模块。其中，网格资源搜索引擎动态周期性地从各制造资源结点搜索可用资源；网格信息交换模块在各类结点进行协同并行或串行工作时，进行各种共享信息的交换和数据转换；资源管理与配置模块对全局的资源进行统一的管理和调度分配，实现资源的共享和高效利用；非技术因素管理模块向制造资源提供方提供注册功能，对客户id提供认证访问全局数据库的功能，同时确定知识产权与利益的分配机制。

③ 安全和可靠性管理。包括网格信息安全和网格资源可靠性两大模块，用于确保数据和信息的完整性、保密性和资源结点的可靠性，为共享资源的管理、调度和优化配置提供依据。

（3）制造网格应用标准（STEP, IGES, STL, RT 相关标准, TS 相关标准, StepML）。在每种类型资源结点的管理 Agent 与通用网络的连接中间，都存在着一定的标准，制造网格应用标准继承了网格技术和通用制造技术的一般特点及其基本标准，并在此基础上，研究关系到制造网格的共享性、通用性和安全性的一些标准，主要包括网络数据传输标准等。

（4）客户端应用程序（应用结点）。在此结点重点开发客户端应用程序，提供友好的用户界面和应用层传输和读取的可视化工具。面向 Internet 中的各种客户，为用户提供三种功能：网格资源搜索、任务提交和工作进程查询。

按照应用模式划分，网格应用可以分成很多不同的类型，应用于不同的应用领域。如航空航天的网络化、区域化，仿生、制药研究，国家天文台的高层应用，制造业、建筑业、科学研究和生物信息等领域应用，以及企业信息系统、电子政务和数字图书馆等。

中科院计算所为自己的网格研究命名为“织女星网格”，这个网格项目的目标是具备以下三种能力：① 大规模的数据处理能力；② 高性能计算能力；③ 具备资源共享和提高资源利用率的能力。“织女星网格”的最大特点是“服务网格”（Service Grid）的概念。而服务网格有三个要点：① 它是一种通用网格，不只支持科学计算，还支持其他服务，包括通信服务、数据服务、信息服务、计算服务、交易服务等；② 服务是基本的应用模式，即客户端向网格发出服务请示，网格完成服务，并将结果通知客户端；③ 网格的主要评价标准不单纯是计算速度等传统指标，而是类似“服务等级协议”（Service Level Agreement）这样一套用户满意度或服务质量评价标准。

11.4 云计算

云计算（Cloud Computing）是一种计算模式或一种商业模式，在学术界包括业界中存有不同意见。赞美者认为，“云计算已经来到，正在和将会改变你我的生活”。反对者认为，“云计算是一个很有煽动性的广告语”。但不管是发展趋势还是商业炒作，云计算成为计算机技术发展上最响亮的名词之一已经是不争的事实。为此，我们收集了一些资料，对云计算给出简介，以便读者对云计算有所了解。

在各种计算机技术架构图中常用一个云团来表示互联网，所以云计算可以理解作为一种基于互联网的计算机模式。显然，云计算谈不上是一种完全崭新的计算机技术，但是用“服务”来包装它，使云计算成为了如此引人兴奋的一种商业模式。

11.4.1 云计算的定义

要给云计算下确切定义或许是困难的，我们选择几种相对比较认可的说法来简要地说明云计算。

（1）**维基（Wikipedia）百科**：云计算是分布式计算技术的一种，其最基本的概念，是通过网络将庞大的计算处理程序自动分拆成无数个较小的子程序，再交由多部服务器所组成的庞大系统经搜寻、计算分析之后将处理结果回传给用户。

（2）**百度百科**：云计算是分布式处理（Distributed Computing）、并行处理（Parallel Computing）和网络计算（Grid Computing）的发展，或者说是这些计算机科学概念的商业实现。云计算的基本原理是，通过使计算分布在大量的分布式计算机上，而非本地计算机或远程服务器中，企业数据中心的运行将更与互联网相似，这使得企业能够将资源切换到需要的应用上，根据需求访问计算机和存储系统。

（3）**伯克利（Berkeley）白皮书**：云计算包含互联网上的应用服务及在数据中心提供这些服务的软/硬件设施。互联网上的应用服务一直被称为软件服务（Software as a Service, SaaS），所以我们使用这个术语。而数据中心的软/硬件设施就是我们所说的云。当云以即用即付的方式提供给公众时，我们称其为“公共云”，这里出售的是效用计算（Utility Computing）。当前典型的效用计算有 Amazon 的 Amazon Web Services、Google 的 Google AppEngine 和 Microsoft 的 Azure 等。不开放的企业或组织内部数据中心的资源称其为“私有云”。因此云计算就是 SaaS 和效用计算，但通常不包括私有云。

我们认为，在云计算模式中，用户所需的应用程序并不运行在用户的个人电脑、手机等终端设备上，而是运行在互联网上大规模的服务器集群中。用户所处理的数据也并不存储在本地，而是保存在互联网上的数据center里。提供云计算服务的企业负责管理和维护这些数据中心的正常运转，保证足够强的计算能力和足够大的存储空间可供用户使用。而用户只需要在任何时间、任何地点，用任何可以连接至互联网的终端设备访问这些服务即可。其目标是以公开的标准和服务为基础，以互联网为中心，提供安全、快速、便捷的数据存储和网络计算服务，让互联网这片“云”成为每一个网民的数据中心和计算中心。

11.4.2 云计算的一个典型例子

在纳斯达克交易所，研究员查尔斯·克劳斯想为客户提供查询历史交易数据的服务并且使信息精确到毫秒。他原本预计将耗资数十万美元去购买服务器等软/硬件设备，但经过比较之后，他选择了亚马逊（Amazon）云计算服务。花费还不到 500 美元就实现了同样的功能。两者之间居然相差 1000 倍以上，这似乎很难理解吧？其实这很好理解，如果你想在家里自己上网，那么需要付出购买计算机和宽带费等至少 3000 美元以上的代价，而你去一个网吧，1 小时假设平均支付 3 美元，也是相差 1000 倍。而“云计算”就是全世界的超级网吧服务。

11.4.3 云计算的特点

（1）云计算提供最可靠、最安全的数据存储中心，用户不用再担心数据丢失、病毒入侵等麻烦。很多人觉得数据只有保存在自己看得见、摸得着的计算机里才最安全，其实不然。用户的计算机可能会因为自己不小心而被损坏，或者被病毒攻击，导致硬盘上的数据无法恢复，而有机会接触用户计算机的不法之徒则可能利用各种机会窃取你的数据。

(2) 云计算对用户端的设备要求最低，使用起来也最方便。大家都有过维护个人计算机上种类繁多的应用软件的经历。为了使用某个最新的操作系统，或使用某个软件的最新版本，我们必须不断升级自己的计算机硬件。为了打开朋友发来的某种格式的文档，我们不得不疯狂寻找并下载某个应用软件。为了防止在下载时引入病毒，我们不得不反复安装杀毒和防火墙软件。

(3) 云计算可以轻松实现不同设备间的数据与应用共享。在云计算的网络应用模式中，数据只有一份，保存在“云”的另一端，用户的所有电子设备只需要连接互联网，就可以同时访问和使用同一份数据。这一切都是在严格的安全管理机制下进行的，只有对数据拥有访问权限的人，才可以使用或与他人分享这份数据。

(4) 云计算为我们使用网络提供了几乎无限多的可能，为存储和管理数据提供了几乎无限多的空间，也为我们完成各类应用提供了几乎无限强大的计算能力。个人计算机或其他电子设备不可能提供无限量的存储空间和计算能力，但在“云”的另一端，由数千台、数万台甚至更多服务器组成的庞大的集群却可以轻易地做到这一点。

11.4.4 云计算的一种实现举例

在 Hadoop 架构上使用 Map/Reduce 技术来实现云计算，如图 11-4 所示。

云计算架构 Hadoop	
Map/Reduce API (Map, Reduce)	BigTable (分布式数据库)
GFS (Google 分布式文件系统)	

图 11-4 云计算的一种实现

在架构中，由 Map/Reduce API 提供 Map 和 Reduce 处理、GFS 分布式文件系统和 BigTable 分布式数据库的数据存取。

Hadoop 是专门负责分布式存储及分布式运算的项目，它是一个更容易开发和处理大规模数据的软件平台。它的主要特点如下：

- (1) 扩容能力。能可靠地存储和处理千兆字节 (PB) 数据。
- (2) 成本低。可以通过普通机器组成的服务器群来分发和处理数据，这些服务器群总计可达数千个结点。
- (3) 高效率。通过分发数据，Hadoop 可以在数据所在的结点上并行地处理它们，且处理非常快速。
- (4) 可靠性。Hadoop 能自动维护数据的多份副本，并且在任务失败后能自动地重新部署计算任务。

Hadoop 实现了一个分布式文件系统 (Hadoop Distributed File System, HDFS)。HDFS 具有高容错特点，并且用于部署在低廉的硬件上。它提供高传输速率来访问应用程序的数据，适合那些有着超大数据集的应用程序。HDFS 还可以“流访问”(Streaming Access) 文件系统中的数据。

Hadoop 实现了 Map/Reduce 分布式计算模型。Map/Reduce 将应用程序的工作分解成很多小的工作块。HDFS 为了实现可靠性，创建了多份数据块的副本，并将它们放置在服务器群

的计算结点中，Map/Reduce就可以在它们所在的结点上处理这些数据了。

HDFS 采用的是典型的主从架构，实现起来相对简单。HDFS 集群有一个 NameNode，它是一个中心服务器，负责管理文件系统的元数据。除了 NameNode，还有一定数量的 DataNode。一般来说一个结点一个 DataNode，负责管理该结点上存储的数据。从外部来看，HDFS 就是一个文件系统，数据以文件的形式存储。在内部，文件被分割成块，存储在多个 DataNode 上。NameNode 执行 namespace 相关的操作，比如打开、关闭以及重命名文件和目录，同时维护文件和块的映射关系，即一个文件有几块，分别存在哪几个 DataNode 上这样的元数据。DataNode 负责对客户端的读/写请求进行服务，同时在 NameNode 的指挥下进行块的创建、删除和复制。NameNode 掌握文件的信息，而 DataNode 面对的是数据块。单个的 NameNode 简化了 HDFS 的设计，NameNode 只存储元数据，用户的数据永远不会传到 NameNode 上。

NameNode 和 DataNode 都是逻辑上的概念，它们是运行在普通的机器上的常驻程序。HDFS 是用 Java 写的，装有 Java 虚拟机的系统都可以运行 NameNode 和 DataNode。

最初由 Google 工程师设计并实现的 Map/Reduce 是一个用于大规模数据处理的分布式计算模型。用户定义一个 Map 函数来处理一个 key/value 对以生成一批中间的 key/value 对，再定义一个 Reduce 函数将所有这些中间的有着相同 key 的 values 合并起来。很多现实世界中的任务都可用这个模型来表达。

Map/Reduce 框架的运作完全基于<key,value>对，即数据的输入是一批<key,value>对，生成的结果也是一批<key,value>对，只是有时候它们的类型不一样而已。key 和 value 类由于需要支持被序列化操作，所以它们必须要实现 Writable 接口，而且 key 类还必须实现 WritableComparable 接口，使框架可以对数据集执行排序操作。

运行于 Hadoop 上的 Map/Reduce 应用程序最基本的组成部分包括一个 Mapper 和一个 Reducer 类，以及一个创建 JobConf 的执行程序，在一些应用中还可以包括一个 Combiner 类，它实际也是 Reducer 的实现。

11.5 性能评价和测量

用户总希望能以最小的代价，最有效地利用系统的一切资源，使之具有尽可能高的信息处理能力，以满足需要。因此用户总是不断追求计算机系统的性能价格比的改善，迫切需要了解对计算机系统进行性能评价的结果。设计人员又要不断地提高计算机系统的性能价格比，不断地对计算机系统的性能进行评价和测量。

计算机性能评价是指计算机系统对原始数据进行逻辑推算。计算机性能测量是指采用基准测试程序包来度量计算机系统的性能。

11.5.1 性能评价的标志

长期以来，计算机系统性能的主要标志是计算机速度。随着计算机系统结构的发展和复杂化，仅从计算机速度来反映计算机系统性能就不合适了。应该对计算机系统的硬件、软件等各个方面进行更为准确的评价，才能全面反映计算机系统的性能。但是计算机系统的速度仍然是衡量计算机系统性能最直接和最主要的标志之一。

计算机系统中 CPU 的主频往往决定了机器周期的长短，从某种意义上说，它能反映计算

机的速度。同样的 PC 机, CPU 的主频可以是 500MHz, 700MHz, 1GHz 等。显然主频越高, 计算机运行的速度就越快。但是如果计算机的组成不同, 则 CPU 的主频就不一定能反映出机器的真正运行速度了。为此, 希望用指令运行速度来表示机器的速度。由于指令种类很多, 且执行时间又不一样, 在程序中各类指令使用频度又存在很大的差异, 因此出现了好多种评价方法。

(1) 常用的计量单位有定点整数指令每秒运算百万次 MIPS (Million Instruction Per Second) 和浮点指令每秒运算百万次 MFLOPS (Million Floating-point instruction Per Second)。

(2) 早期采用加法指令的运算速度来标志机器的速度。因为那时指令系统的指令种类少, 且大致运算速度差距不大, 加法指令又是指令系统中使用频度很高的指令, 有一定的代表性。

(3) 随着指令系统的发展, 仅用加法指令运算速度已不能很好地反映机器速度, 出现了“等效指令速度法”。它通过取各类指令在程序中的比例 (ω_i) 进行折算来获得速度值。若每类指令的执行时间为 t_i , 则等效指令的执行时间 T 为

$$T = \sum \omega_i t_i$$

机器的等效指令速度

$$v = 1/T$$

(4) 等效指令速度法比例 (ω_i) 选取的程序不同应用场合和程序本身性质不同, 会有很大差别。同时它又忽视了软件、外部设备等其他方面, 不能全面反映机器的速度。现在性能评测都采用“基准测试程序法”标准, 如用 SPEC Mark, LINPACK 基准测试程序等。选用最频繁使用的核心程序段、包括编译、I/O 操作等的典型程序段, 甚至人为地编制能涉及各方面操作的考核程序, 将它们混合起来作为评价机器速度的标志。

尽管这样, 要满意地、完全确切地反映机器速度的真实性还是有一定难度的。

11.5.2 性能的描述

计算机系统的性能主要反映了一个系统的使用价值, 即性能价格比。广义的性能含义包括系统处理能力、响应速度、工作效率、可靠性、可使用性、可维护性等。这些性能既有可定量的指标, 也有不可定量的指标; 既有可客观测定的, 也有很大程度上取决于评价者的主观性的; 而且与系统的应用环境、工作负载、求解问题规模的大小密切相关; 当然系统的价格是决不能忽视的重要因素。所以在讨论计算机系统的性能时不能脱离实际条件。

11.5.3 性能评价的对象

性能评价的对象是整个计算机系统。但计算机系统是包括硬件、软件的复杂系统, 又与工作环境、工作方式、应用对象等有密切的关联, 所以要明确地划清计算机系统的环境 (边界环境), 其中最主要的是工作负载。工作负载是加到系统上的服务需求量, 它可以是一个作业对 CPU 工作时间的需求量、存储空间的需求量、I/O 工作的需求量以及软件资源的需求量等。工作负载模型可以是自然负载模型, 或研究者建立的负载模型, 而且精确建立的负载模型可以节约评价时间和开销。实际上, 性能评价最困难的工作是选定有代表性的工作负载模型。

11.5.4 性能评价的手段

在选定工作负载后, 要利用评价手段收集有关系统性能的数据, 这就要用评价技术来实

现。评价技术主要有测量技术（有实际系统存在，并可从系统直接测得数据）和模型技术（只能从模型来测得数据）。

性能测量的主要任务是测试内容的确定；测量工具的选择；测量实验的设计；测量结果的解释和结果的可视化表示。

模型技术又可分为模拟模型和数学分析模型。由于模型技术需要某种假设条件，从模型上获得的数据可信度取决于模型的精确度，而且有待于到真实系统上去验证。模型技术的主要任务是形式化描述模型；实现模型；设计模拟实验；正确性校验；执行实验并测得和分析数据。在分析中最常用的是排队模型。

11.5.5 性能的评价

究竟怎样的系统结构是最好的？一般来说，系统结构的性能不应只以程序执行速度来度量，而应该用实际解题效率来度量。但是前者容易度量且可量化，而后者很难度量，因此前者被广泛地使用和交流。在专业性讨论时，往往用后者更能说明问题。例如，中科院的联想 LSSC-II 机群系统运行国际标准的 Linpack 基准程序 HPL，使用 512 个处理器求解 153600 阶线性方程组，实测性能达到每秒 1.027 万亿次浮点运算。

11.5.6 性能评测标准举例

工业界对计算机系统性能评测比较多的是采用“综合基准测试程序”标准，它是从实际程序中抽取少量关键循环程序段，并考虑了各种操作和各种程序的比例。比较典型的有 Whetstone, Dhrystone 等，而 Linpack 则是一些 FORTRAN 子程序的集合，用来分析和求解线性方程组和线性最小二乘法问题，它是衡量一个系统数值计算能力的良好指示器。

这里要介绍的是应用广泛的由 SPEC (Standard Performance Evaluation Corporation) 公司开发的 SPEC 基准程序系列。它强调开发实际应用基准程序以求更确切地反映实际工作负载，对每个基准程序和基准程序组都定义了少量的指标以测量整个系统的总性能。SPEC 以 CPU 性能的基准程序作为出发点，现已向 Client/Server 计算、商业应用以及 I/O 子系统等方面发展。

SPEC 主要的基准程序组有：

(1) SPEC 95。测量 CPU 存储器系统和编译器代码生成性能。

(2) SPEC hpc 96。测量运行工业型应用程序的高性能计算系统的性能。有两个典型基准测试程序：SPEC seis 96（地震处理基准程序）和 SPEC chem 96（计算化学基准程序）。

(3) SPEC web 96。测量基于有 Web 服务器运行记录而获得的工作负载的 Web 服务器性能。

(4) SFS。为系统级文件服务器基准程序，用于测量在不同负载情况下 NFS 文件服务器的响应时间和吞吐率。

(5) SDM。为系统开发多任务基准程序，测量一个系统处理有大量用户发出典型 UNIX 软件开发命令的环境性能。

(6) GPC。图形性能特征描述基准程序，测量图形学性能。它有三个典型基准测试程序：测量图形处理能力的 PLB (Picture Level Benchmark)、测量 X 窗口性能的 Xmark 93 和测量 Open GL 的 OPC (Open GL Performance Characterization)。

参 考 文 献

- [1] 金兰等. 并行处理计算机结构. 北京: 国防工业出版社, 1982.
- [2] 苏东庄. 计算机系统结构. 西安: 西安电子科技大学出版社, 1984.
- [3] 李学干. 计算机系统结构(第二版). 西安: 西安电子科技大学出版社, 1996.
- [4] 李勇等. 计算机体系结构. 长沙: 国防科技大学出版社, 1988.
- [5] 郑纬民等. 计算机系统结构(第一版). 北京: 清华大学出版社, 1992.
- [6] 郑纬民等. 计算机系统结构(第二版). 北京: 清华大学出版社, 1998.
- [7] 李三立等. RISC单发射与多发射体系结构. 北京: 清华大学出版社, 1993.
- [8] 陆鑫达. 计算机系统结构. 北京: 高等教育出版社, 1996.
- [9] 张吉锋等. 计算机系统结构. 北京: 电子工业出版社, 1997.
- [10] 徐炜民等. 计算机系统结构(第二版). 北京: 电子工业出版社, 2003.
- [11] 黄铠等. 可扩展并行计算——技术、结构与编程. 北京: 机械工业出版社, 2000.
- [12] 张昆藏. 计算机系统结构教程. 北京: 国防工业出版社, 2001.
- [13] 都志辉等. 网格计算. 北京: 清华大学出版社, 2002.
- [14] Kai Hwang.Advanced Computer Architecture—Parallelism Scalability Programmability.McGraw-Hill,Inc, 1993.
- [15] 王鼎兴等译. 高等计算机系统结构——并行性、可扩展性、可编程性. 北京: 清华大学出版社, 1995.
- [16] William Stallings. Computer Organization and Architecture—Design for Performance. Prentice Hill, 1996.
- [17] Kai Hwang, Zhiwei Xu.Scalable Parallel Computing—Technology, Architecture, Programming. McGraw-Hill,Inc, 1999.
- [18] 陆鑫达等译. 可扩展并行计算—技术、结构与编程. 北京: 机械工业出版社, 2000.
- [19] 郑纬民等译. 高性能集群计算: 结构与系统(第一卷). 北京: 电子工业出版社, 2001.
- [20] 郑纬民等译. 高性能集群计算: 编程与应用(第二卷). 北京: 电子工业出版社, 2001.
- [21] John L. Hennessy, David A. Patterson.Computer Architecture—A Quantitative Approach (Third Edition).Morgan Kaufmann, 2002.
- [22] 邹恒明等译. 计算机系统设计与结构(第二版). 北京: 电子工业出版社, 2005.
- [23] Rob Williams.Computer Systems Architecture—A Networking Approsch (Second Edition).Pearson Education, 2006.
- [24] 赵学良等译. 计算机系统结构(第二版). 北京: 机械工业出版社, 2008.